UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE MEDIDA DE VELOCIDAD PARA PRUEBAS DE ATLETISMO.

Juan José Huerta Cristóbal

Tutor: Clara Pérez Fuster

Cotutor: Fulgencio Montilla Meoro

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2019-20

Valencia, 10 de septiembre de 2020

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO. ES-CV- 000030

Resumen

El trabajo que se presenta consiste en el desarrollo e implementación de un sistema inalámbrico de medida de tiempos aplicado a las carreras de atletismo. El sistema se basa en sensores ópticos conectados a microcontroladores, concretamente Arduino, a los que se les ha incorporado módulos de radio frecuencia y Bluetooth, para la comunicación entre ellos y con un dispositivo móvil para facilitar al deportista el manejo del equipo. La alimentación del equipo será mediante baterías externas, permitiendo su uso en pistas de entrenamiento donde no se dispone de puntos de alimentación eléctrica; a la vez que lo hace portable a cualquier lugar.

Mediante este equipo se quiere medir el tiempo que tarda un atleta en recorrer una determinada distancia y deducir su velocidad media durante la carrera. La configuración de los diferentes parámetros de la prueba de velocidad, así como el almacenamiento de los tiempos realizado se llevará a cabo mediante una aplicación instalada en un terminal móvil con sistema operativo Android, que permitirá al corredor controlar el equipo.

Resum

El treball que es presenta consistix en el desenvolupament i implementació d'un sistema sense fil de mesura de temps aplicats a les carreres d'atletisme. El sistema es basa en sensors òptics connectats a microcontroladors, concretament Arduino, als que se'ls 'ha incorporat mòduls de ràdio freqüència i Bluetooth, per a la comunicació entre ells i amb un dispositiu mòbil per a facilitar a l'esportista el maneig de l'equip. L'alimentació de l'equip serà a través de bateries externes, permetent el seu ús en pistes d'entrenament on no es disposa de punts d'alimentació elèctrica, al mateix temps que ho fa portable a qualsevol lloc.

Per mitjà d'aquest equip es vol mesurar el temps que tarda un atleta en recórrer una determinada distància i deduir la seua velocitat mitjana durant la carrera. La configuració dels diferents paràmetres de la prova de velocitat, així com l'emmagatzemament dels temps realitzats, es durà a terme a través d'una aplicació instal·lada en un terminal mòbil amb sistema operatiu Android, que permetrà al corredor controla l'equip.

Abstract

The aim of this project is to develop and implement a wireless time measuring system applied to athletics races. The system is optic-scanners based, connected to microcontrollers (specifically Arduino type) to which radio frequency and Bluetooth modules have been incorporated, in order to communicate between them and with a mobile device, providing to the athlete an easy utilization of the equipment.

Equipment will be power feed by external batteries, allowing it to be used in running tracks where no electrical supply can be found. Therefore, the resulting system becomes portable and can be deployed anywhere.

By means of this solution, the purpose is to measure the time an athlete needs to cover certain distance, extracting the average speed while racing. The configuration of the different parameters of the speed test, as well as the storage of the performed marks, will be done though an App installed in a mobile phone with Android OS, which will allow the athlete to use the device.

Índice

1. Introducción y Antecedentes

1.1 Introducción y Antecedentes

Actualmente vivimos en una era en que la tecnología forma parte de nuestro a día a día, desde la hora de cocinar nuestros alimentos mediante aparatos electrónicos como los microondas, a la fabricación de robots que son capaces de comportarse físicamente como un humano. En el ámbito deportivo se pueden encontrar innumerables sistemas de mejora para el rendimiento físico de los deportistas, tales como pueden ser los relojes deportivos con medidor de pulsaciones, oxígeno en sangre o sistemas de entrenamiento profesional que miden la velocidad y la aceleración en la ejecución de un movimiento de halterofilia.

Estos avances son de gran importancia para el entrenamiento y mejora de los deportistas en el ámbito profesional, dejando muchas veces de lado el sector amateur o aficionado debido al alto precio de estos productos; es ahí donde nace la idea de este proyecto con él se pretende cubrir las necesidades de un sector que muestra gran interés, y no pueden permitirse el coste de los equipos comerciales actuales.

La tecnología actual nos ha acostumbrado a que el manejo de app para dispositivos móviles sea normal en nuestra vida cotidiana; es por ello que el control del equipo de medida mediante una app para móvil, no requeriría una preparación técnica del usuario. Por este motivo y con el fin de presentar un producto elaborado y de fácil uso, el diseño del equipo ha requerido dividir el trabajo en dos partes, una parte dedicada al desarrollo e implementación del sistema físico y la otra parte centrada en el desarrollo de la app; siendo indispensable una parte para la otra para poder realizar una integración perfecta entre ambas; lo cual hace que se trate de un proyecto más real y que requiere de la colaboración entre los desarrolladores, siendo necesaria una comunicación continua entre ambos; es decir, un trabajo en equipo para ir comprobando el ensamblaje de las etapas conforme se avanza en el diseño hasta su finalización y puesta a punto del equipo diseñado.

Este trabajo se dedicará exclusivamente al desarrollo e implementación del sistema físico, comprobando que éste responde correctamente a las instrucciones dadas por el usuario a través de la aplicación desarrollada en el trabajo complementario.

1.2 Motivación

Como corredor de montaña aficionado a la hora de hacer entrenamientos en pista y poder medir el tiempo empleado, siempre me he visto limitado a los clásicos relojes con *GPS*, válidos en distancias largas; pero poco fiables en pistas de atletismo. La otra opción era contar con un compañero de entrenamiento el cual midiese con un cronometro el tiempo que tardaba en recorrer una determinada distancia. Estas limitaciones vienen impuestas por el elevado precio de los equipos profesionales, así como por sus limitaciones a la hora de adaptarse a las nuevas tecnologías, como pueden ser las aplicaciones de los dispositivos móviles. Con esta aplicación se pretende integrar las nuevas tecnologías al ámbito deportivo y abaratar el coste de los equipos de medidas, para que dejen de ser de uso exclusivo y solo aptos para atletas profesionales, clubs importantes o atletas aficionados con un poder adquisitivo alto. Siendo la motivación principal implementar un sistema de medida diseñado por mí mismo, con el cual pudiese llegar a un público mayor y poder realizar mis entrenamientos con mayor fiabilidad, a partir de los conocimientos adquiridos durante mis estudios del Grado de Ingeniería de Sistemas y Servicios de Telecomunicación.

1.3 Descripción del equipo de medida de velocidad

El equipo de medida de velocidad, se diseñará como Trabajo Final de Grado junto de otro compañero, siendo el resultado de la unión de ambos trabajos una colaboración conjunta que permitirá desarrollar un equipo y que su control sea mediante una aplicación móvil. La integración del sistema físico con la aplicación requiere una gran compenetración entre ambos TFG's, lo que exigirá aprender a trabajar en equipo, siendo ésta una competencia muy necesaria actualmente, y a la vez servirá para poner en práctica los conceptos estudiados en la carrera.

El sistema estará basado en microcontroladores del tipo Arduino, conectado a un conjunto de" puertas" inalámbricas, una de ellas llamada" Master" y formada por un Arduino UNO, un módulo Bluetooth HC-05 y un módulo transceptor inalámbricos, el resto de puertas llamadas" esclavos" estarán compuestas por Arduinos NANO y módulos transceptores inalámbricos que recibirán las señales de los sensores.

La puerta master es la encargada de comunicarse mediante tecnología bluetooth con la aplicación móvil recibiendo la señal de START con un mensaje con el tiempo de inicio. Seguidamente cuando la carrera comience y el atleta pase por la primera puerta (Master) cortando el haz de luz emitido por la fotocélula reflexiva el Arduino será el encargado de tomar la medida de tiempo y guardarla para su posterior envió a la aplicación móvil, cuando se corten las siguientes puertas(esclavas) estas enviaran esa medida de tiempo mediante radiofrecuencia al Arduino master y este será el encargado de transmitir los tiempos a la aplicación móvil mediante bluetooth.

Se trata de un sistema dotado de autonomía mediante baterías, lo cual hace que las puertas puedan ser usadas sin necesidad de estar conectadas a la red eléctrica ya que en ocasiones esto puede ser un impedimento para la libre colocación de las mismas

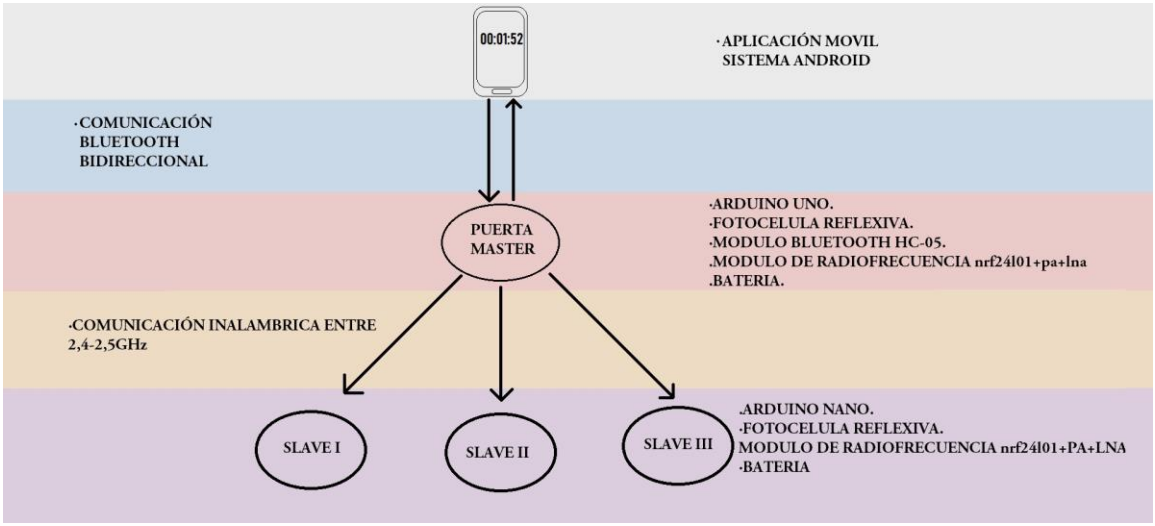

Figura 1. Esquema de la jerarquía del proyecto.

2. Justificación y Objetivos

2.1. *Justificación.*

Las justificaciones del presente trabajo son dos: la primera es académica, y es la obtención del título de Grado en Ingeniería y Servicios de las Telecomunicaciones; la segunda es tal como he comentado en la motivación, la de aportar una mejora técnica a la sociedad del mundo deportivo, en concreto el desarrollo del equipo de medida de velocidad controlado con el dispositivo móvil; y social porque se pretende abaratar un producto existente en el mercado y así ponerlo al alcance de los deportistas amateurs.

2.2. *Objetivos.*

El objetivo principal de este proyecto es desarrollar un sistema de medida de velocidad en carreras de atletismo, portátil, inalámbrico y alimentado por baterías.

A continuación, se presentan los objetivos específicos que deberá cumplir la aplicación:

- Se podrán modificar las características de la prueba, tales como la distancia que se pretende correr.
- Detectará si los sensores están bien posicionados para calibrar el equipo antes de empezar la prueba.
- Deberá poder gestionar el encendido y apagado de las fotocélulas para evitar consumos de batería innecesarios y alargar su duración.
- Se espera que el equipo pueda usarse sin tener conexión a internet.
- Sera posible conocer el estado en el que se encuentra el dispositivo.
- Emitirá algún tipo de señal (sonora o visual) cuando un corredor corte la puerta.
- Su funcionamiento deberá ser exclusivamente inalámbrico.

3. Diseño del Sistema Hardware.

El sistema Hardware consta de las siguientes partes diferenciadas: el subsistema microcontrolador formado por los Arduinos; los sensores ópticos conectados a ellos, que detectarán el inicio y el final de la carrera, así como puntos intermedios; y finalmente el subsistema de Alimentación de todo el equipo, tanto de los Arduino, como de los sensores ópticos.

La descripción de cada parte, junto con su funcionamiento los diferentes sensores y dispositivos de comunicación requeridos por el sistema, se detallarán en los apartados siguientes.

- Subsistema microcontrolador: Arduino UNO y Arduino NANO
- Sensores ópticos
- Subsistema de Alimentación

3.1.Subsistema microcontrolador.

Arduino es una plataforma de creación de prototipos de código abierto basada en el uso sencillo de hardware y software.

Arduino consta de una placa de circuito programable física (microcontrolador) y una parte de software, o IDE (entorno de desarrollo integrado), que se utiliza para diseñar y cargar código de ordenador en la placa física. Estas placas se diseñan con la intención de poder llevar a cabo proyectos de electrónica, está dotada de entradas capaces de leer señales como por ejemplo la activación de un pulsador y mediante el desarrollo de código poder transformarlas en salidas como puede ser el encendido de una bombilla.

Actualmente en el mercado existen gran variedad de modelos de placas Arduino como Arduino NANO, Uno, Due, etc. así como un abanico de fabricantes siendo los más importantes Arduino y ELEGOO.

La parte de Hardware está compuesta por una PCB junto a un microcontrolador, el modelo de este variara en función del tipo de Arduino, usualmente se usa un microcontrolador Atmel AVR (8bit), una serie de puertos analógicos y digitales de entrada y salida que permiten la interacción con otros componentes electrónicos (placas Shield, sensores, leds..).

Por otro lado, la parte software se caracteriza por su simplicidad, esta se basa en un entorno de desarrollo IDE (integrated development environment) con un lenguaje inspirado en Wiring y el cual se basa en un entorno Wiring siendo el bootloader su cargador de arranque encargado de la ejecución en placa. Para la programación del microcontrolador es necesario un ordenador y mediante el uso de una comunicación serial y un convertido de niveles RS-232 a TTL serial.

Para nuestro proyecto se ha hecho uso de placas Arduino UNO y Arduino NANO de la marca ELEGOO debido a su relación calidad-precio.

3.1.1 Arduino UNO R3.

El Arduino UNO R3 va a ser la parte más importante del sistema, encargada de mantener comunicado el equipo de medida con la aplicación móvil. Como se ha comentado en el apartado anterior, existe un gran número de placas Arduino con diferentes características en cuanto a tamaño, memoria, número de puertos etc... En este caso se ha elegido un Arduino UNO R3 y no otro como el Arduino MEGA por diferentes motivos, el principal es el número de pines analógicos de los que dispone siendo mucho menor que su hermano mayor; pero los suficientes para cumplir con todas las funciones que se desean realizar, por otro lado, presenta un tamaño, mucho menor lo que lo hace idóneo para este proyecto y por ultimo su precio, el cual es significativamente inferior a otros, lo que permite abaratar los costes notablemente.

Esa placa de microcontrolador basada en un ATmega328P, concretamente usa un microcontrolador ATmega16U2, el cual dispone de una mayor memoria y un ratio de transferencia superior, frente al ATmega8u2. Esta placa cuenta con todo lo necesario para su funcionamiento sin tener que añadirle elementos externos. Se conecta a un ordenador mediante un cable USB A/B por el cual se alimentará y se cargará el código deseado para su configuración.

Arduino UNO R cuenta con 14 pines digitales de entrada/salida de los cuales 6 se podrán utilizar como salidas PWM (para lectura de valores comprendidos entre 0 y 255.). cuenta con un oscilador de cristal de 16MHz.

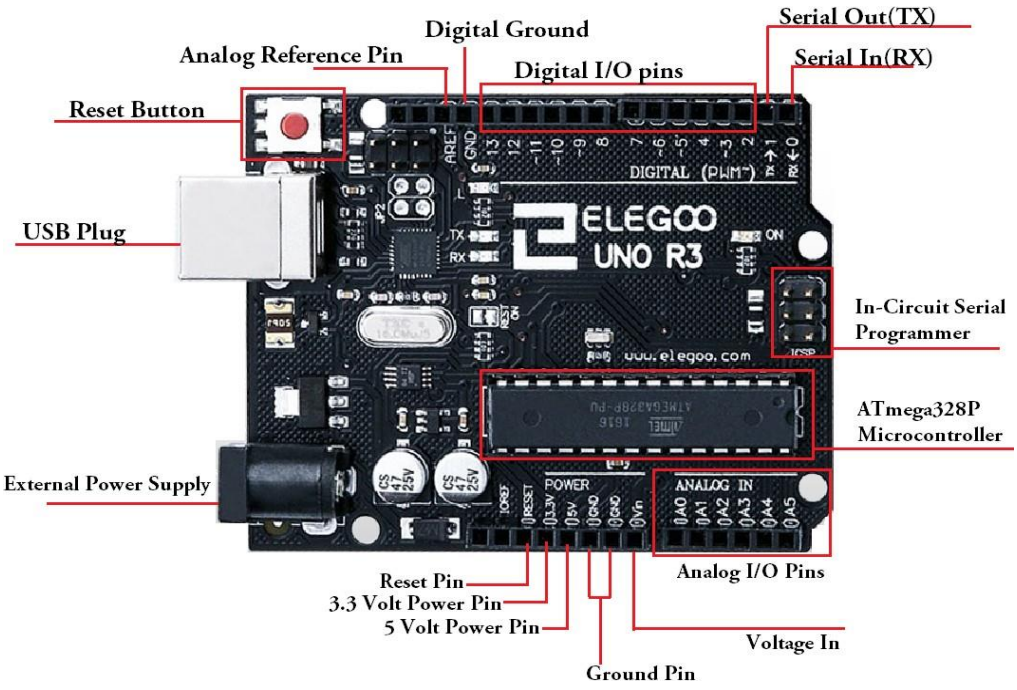

Figura 3.1 PinOut Arduino UNO

Características:

- Microcontrolador**:** ATmega328P.
- Voltaje de entrada: 7-12V.
- Voltaje operativo: 5V.
- Corriente DC por pin de E/S:40mA
- 14 pines de I/O digitales (6 salidas PWM).
- 8 entradas analógicas.
- Memoria Flash de 32k.
- 2 KB de SRAM.
- 1 KB de EEPROM.
- 16Mhz Clock Speed.

3.1.1.1. Entradas y Salidas.

Arduino UNO R3 dispone de 14 pines digitales los cuales pueden ser usados como entradas o salidas según se quiera mediante el uso de las funciones *digitalRead()* para poder leer el valor del pin(entrada), *digitalWrite()* para dar valor a un pin(salida) o *pinMode()* con la cual se configurara el pin seleccionado si se desea que se comporte como entrada o salida. Estos 14 pines operan a una tensión de 5 Voltios y pueden proporcionar o recibir una intensidad máxima de 40mA poseyendo y una resistencia interna(pull-up) de entre 20-50KΩ la cual viene desactivada por defecto.

Por otro lado, se dispone de 6 pines Analógicos que también pueden ser configurados como entrada o salida los cuales proporcionan una resolución de 10bits la cual se mide por defecto desde 0V a 5V siendo posible modificar este último valor mediante la función *analogReference()* y el pin AREF.

El resto de pines importantes que se van a usar disponen de otras funciones como son:

- **PWM [3,5,6,9,10,11**]: Los pines PWM(Pulse With Modulation) son usados para la modulación por anchura de pulso a 8bits(256 valores) mediante la función analogWrite().

- **Serial Pins [0(RX), 1(TX)]:** Estos pines son usados para recibir y transmitir datos en serie TTL en la comunicación con dispositivos externos. Cada uno de ellos están conectados a sus respectivos pines del chip USB-to-Serial ATmega16U2.

- **AREF:** AREF significa Referencia analógica. Permite cambiar la tensión de referencia de los pines analógicos del Arduino conectando una fuente de alimentación externa a dicho pin. Por ejemplo, si se quiere medir voltajes con un rango máximo de 3.3V, se alimentaria con 3.3V en el pin AREF.

- **RESET: C**uando se le Suministra un valor LOW (0V) se reinicia el microcontrolador y es utilizado normalmente para añadir un botón externo que ejecute dicha acción.
- **SPI (10(SS), 11(MOSI), 12(MISO), 13(SCK)):** Pines los cuales proporcionan comunicación síncrona mediante el protocolo SPI usando la librería SPI.
- **Analog Pin (A0-A6):** pines de entrada analógicos los cuales digitalizan las señales de entrada con una resolución de 10 bits, estos pines trabajan en un rango de 0-5V, este rango se puede modificar mediante el pin AREF y la función *AnalogReference()*.

3.1.1.2. Alimentación.

La alimentación del Arduino UNO se hace usualmente mediante conexión USB mediante el cual también se carga el código a ejecutar, la placa Arduino también dispone de un conector Jack de 2.1mm y un pin de entrada V_{in}. La selección del método de alimentación se hace automáticamente al conectarla ya sea vía USB, Jack o pin V_{in}.

La placa Arduino UNO trabaja a un voltaje de +5V, admitiendo voltajes comprendidos de entre 7-12V, voltajes mayores a los mencionados pueden provocar el sobrecalentamiento del regulador de voltaje. Los pines I/O de alimentación, así como los diferentes conectores más importantes de los que dispone esta placa Arduino UNO y sus características son los siguientes:

- **Vin:** La conexión de una fuente de alimentación sobre este pin y GND permite alimentar a la placa en un rango de entre 6-12v. Este método de alimentación de la placa no cuenta con ningún dispositivo de protección contra inversión de polaridad, así como tampoco contra subidas de tensión. Cuando el Arduino esté alimentado mediante la conexión Jack este pin sirve como salida de voltaje, brindando un voltaje de salida igual al que se esté aplicando en la entrada Jack.
- **Pin 5V:** Este pin puede funcionar como una salida estable de +5V para circuitos externos o como entrada para alimentar Arduino cuando no hay un cable USB conectado o un adaptador de corriente Jack. No se dispone de protección si se decide alimentar el Arduino mediante este pin.
- **Jack:** Entrada por la cual se puede alimentar la placa Arduino, normalmente mediante un adaptador de corriente (AC/DC) con un voltaje adecuado entre 7-12V DC. Si se aplica voltajes menores a 7V pueden provocar que el regulador interno de Arduino no funcione correctamente. Esta entrada dispone de diodos de protección frente a inversiones de polaridad, así como a subidas de tensión.
- **Pin 3.3V:** Este pin suministra un voltaje de salida de 3.3V y una intensidad de 50mA generado por el regulador de tensión interno de Arduino.

3.1.1.3. Memoria.

El Arduino UNO dispone de un microprocesador AT- mega328, el cual cuenta con una memoria de 32 kB, de la cual se reservan 0.5 kB para el bootlader (gestor de arranque) y los 31.5 kB restante están dedicados a almacenar el código del programa. También incluye una EEPROM de 1 kB y una SRAM de 2 kB.

3.1.1.4. Comunicación.

La comunicación en Arduino es uno de los aspectos más interesantes a la hora de realizar proyectos ya que permite poder comunicar el dispositivo con otros Arduinos mediante diversas tecnologías (bluetooth, Radiofrecuencia...), con un ordenador, otro microcontrolador o incluso con dispositivos móviles.

Los puertos de comunicaciones serie de los que esta placa esta provista, son:

- **I2C:** Es un protocolo de comunicaciones para una interfaz de dos cables, el cual brinda la oportunidad de conectar dispositivos de baja velocidad (microcontroladores, EEPROM...). Este protocolo permite la conexión de varios ”Esclavos” con un ”Maestro” a la placa Arduino, mientras se mantiene una vía de comunicación clara gracias a la utilización de un sistema de direcciones y un bus compartido (dos cables), lo que da la opción de que varios dispositivos se conecten al mismo cable y Arduino ”llame” a cada dispositivo a través de la dirección que se le asigne.

- **SPI:** SPI viene de las siglas ”Serial Peripheral Interface” este protocolo permite un único ”Maestro” y un máximo de cuatro ”Esclavos”, sus líneas de datos/reloj son compartidas entre dispositivos; pero a diferencia del protocolo I2C cada ”esclavo” requiere un cable de dirección, lo que lo hace más rápido.

- **UART:** Las siglas UART significan recepción y transmisión asíncronas universales (Universal Asynchronous Receiver/Transmitter) y es un protocolo de comunicación simple, que permite que el Arduino se comunique con dispositivos serie. El sistema UART usa los pines digitales 0 (RX) y 1 (TX) y con otro PC a través del puerto USB. La conexión Arduino-PC es gracias a que Arduino dispone de un convertidor USB-Serie incorporado (ATmega16U29). Este protocolo es el más usado en la gran mayoría de proyectos, mediante la biblioteca *SoftwareSerial;* permite que cualquiera de los pines de Arduino funcione como bus de datos serie; se debe asignar un pin RX, el cual actuará como receptor y un Pin TX que actuará como emisor. Los datos enviados y recibidos pueden ser mostrados en el ordenador mediante el IDE, el cual está disponible en Arduino, llamado ”Monitor Serial”. También se puede usar esta herramienta para mandar comandos a Arduino.

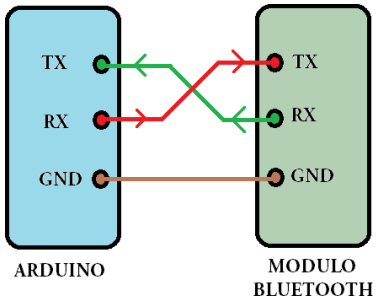

Figura 3.2 Esquema de conexión UART entre Arduino y un módulo Bluetooth

3.1.2. Arduino NANO

El modelo que se utilizará para el control de las puertas ”esclavo” será Arduino NANO; la elección de este modelo de Arduino es clara, su tamaño es el más pequeño de la familia Arduino, con unas medidas de 18 x 45 mm, siendo perfecto para cualquier proyecto en el que se requiera un tamaño lo más compacto posible. El número de pines, tanto digitales como analógicos, es más que suficiente para este proyecto. Este modelo está basado en el microcontrolador MCU Atmel ATmega328p y su mayor característica es el reducido tamaño de la placa y un menor consumo.

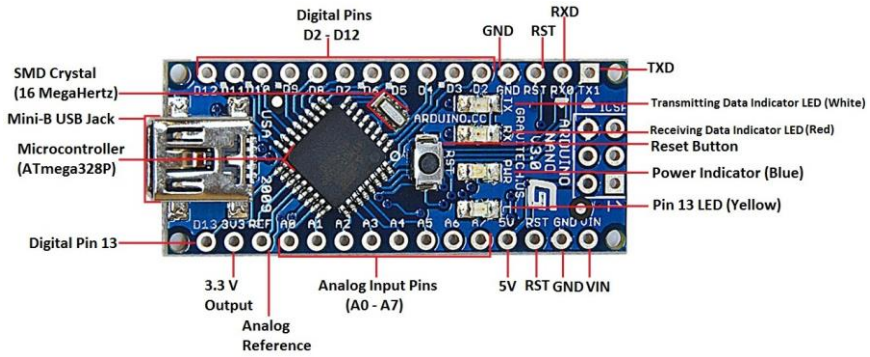

Figura 3.3: PinOut Arduino NANO

- Microcontrolador: ATmega328.
- Arquitectura AVR.
- Voltaje de Funcionamiento: 5V.
- Memoria flash,32 KB de los cuales 2 KB utilizados por bootloader.
- SRAM Velocidad del reloj 16 MHz.
- 8 pines de E/S analógicas.
- EEPROM,1 KB.

- Corriente continua por pin entrada salida, 40 mA (Pines de E/S).
- Voltaje de entrada, 7-12 V.
- 22 pines de E/S digitales.
- 6 salidas PWM.
- Consumo de energía, 19 mA.
- Tamaño de la placa de circuito impreso, 18*45mm.

3.1.2.1. Entradas y Salidas.

Arduino NANO al igual que su hermano mayor Arduino UNO dispone de 14 pines digitales que operan a 5V y los cuales pueden ser configurados como entrada o salida mediante las funciones *pinMode()*, *digitalRead()* y *digitalWrite()*. Cada uno de estos pines puede recibir o suministrar una intensidad máxima de 40mA, poseen una resistencia interna(pull-up) que oscila entre los 20-50KΩ la cual viene desactivada por defecto. Esta placa electrónica dispone de 8 entradas analógicas las cuales operan con una resolución de 10bits (1024 valores), por defecto estos pines miden entre 5V y GND. Como en su hermano mayor es posible cambiar este valor mediante la función *analogReference()*

3.1.2.2. Alimentación**.**

Las diferentes opciones de alimentación de Arduino NANO se simplifican a los pines de V_{in} el cual admite una tensión de entrada de entre 7-12V y una entrada Mini-B USB Jack.

3.1.2.3. Memoria.

El ATmega 328 posee 32kB, de los cuales 2 kB son usados por el bootloader. Posee también 2 kB de SRAM y 1kB de EEPROM.

3.1.3. IDE.

Las siglas IDE vienen de Integrated Development Environment o entorno de desarrollo integrado, se trata de una herramienta la cual permite desarrollar y modificar el código (también llamado Sketch), a la vez que poder comprobar que no contiene errores para posteriormente poder subir el Skecth al microcontrolador.

El lenguaje en el cual se programa el Arduino no es un lenguaje concreto como tal, sino el conjunto de instrucciones provenientes de los lenguajes de programación C y C++, el cual está diseñado para poder simplificar la programación de microcontroladores del tipo AVR.

El IDE oficial de Arduino se caracteriza entre otras cosas por su simplicidad a la hora de usar y su interfaz intuitiva, la cual se muestra en la siguiente figura.

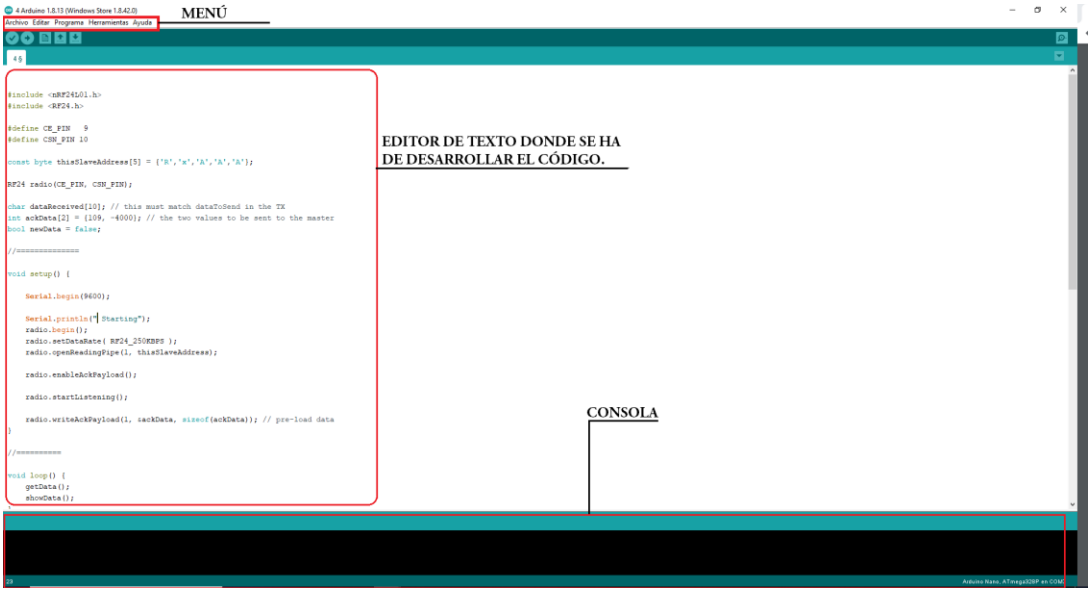

Figura 3.4 Interfaz Arduino IDE.

3.1.4. Comunicación Bluetooth

El termino Bluetooth describe una tecnología bajo el estándar IEEE 802.15.1, es una tecnología de comunicación inalámbrica de corto alcance la cual sirve para la transferencia de datos y voz de punto a punto consiguiendo así sustituir las conexiones por cable. El bluetooth utiliza la banda de ISM de uso no regulado a una frecuencia de 2.4 GHz.

Existen diferentes tecnologías Bluetooth las cuales se clasifican según su potencia/velocidad de transmisión que ha ido aumentando según se mejoraba la tecnología, a continuación, se muestra una tabla donde recoge las diferentes versiones disponibles actualmente:

Bluetooth			
Versión	**Lanzamiento**	**Velocidad de transmisión (Mbit/s)**	**Alcance PAN(m)**
2.0 + EDR	2004	2.1	10
2.1 + EDR	2007	3	10
3.0+HS	2009	24	10
4.0+LE	2010	32	10
4.1	2013	32	10
4.2	2014	32	10
5.0	2017	50	40

Tabla 3.1: Versiones de tecnología Bluetooth

3.1.4.1. Modulo Bluetooth HC-05.

Uno de los objetivos marcados de este proyecto era la necesidad de una comunicación inalámbrica, tanto entre las puertas "Master" y "esclavo", como la comunicación entre la puerta "Master" y el dispositivo móvil al que se conectará. Para esta última conexión se ha elegido una comunicación bluetooth la cual aparte de ser inalámbrica y fiable, también cumple otro de los objetivos, el de poder hacer funcionar el módulo sin conexión a internet.

En el mercado destacan dos módulos bluetooth para Arduino, el módulo HC-06 y el HC-05, la principal diferencia entre estos dos módulos es que este último puede ser configurado para ser usado como esclavo o maestro, presentando mayor versatilidad para aplicar diferentes configuraciones o futuras mejoras, por el contrario, el módulo HC-06 solo puede ser configurado como esclavo.

El módulo HC-05 será el elemento principal de la comunicación bluetooth entre nuestra la *Master* y la aplicación Android. En concreto el módulo bluetooth del que se hace uso en este proyecto usa la tecnología Bluetooth 2.0; es cierto que es una tecnología algo desfasada y por lo tanto su velocidad de transmisión es menor a la de otros dispositivos con tecnología superior, pero esta elección viene dada principalmente por el precio, y porque la cantidad de datos a transmitir no es elevada.

El módulo trabaja a una tensión de alimentación de entre 3.3V y 6V, siendo ideal alimentarlo a 3.3V debido a que sus pines de transmisión (TX y RX) trabajan a dicho voltaje. Este módulo usa una tecnología Bluetooth V2.0, la cual tiene una velocidad de transmisión de 2.1 Mbit/s, su potencia de transmisión es < 4 dBm utilizando una modulación GSFK.

El módulo Bluetooth usado dispone de 6 pines de los cuales solo se usarán 4. Los dos pines de alimentación GND y V_{CC}, dos pines para la transmisión TX y RX y los pines STATE y EN que son los que se utilizarán. También dispone de un pulsador el cual permitirá entrar al modo configuración.

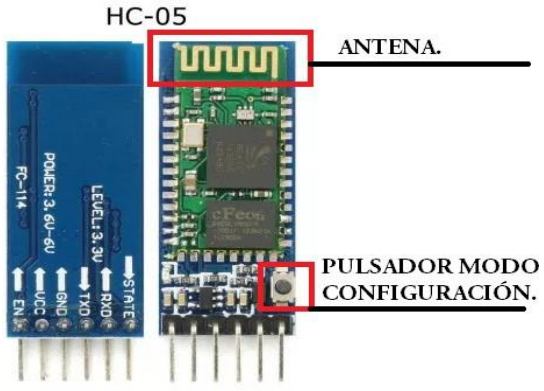

Figura 3.5 Modulo Bluetooth HC-05.

3.1.4.2. Configuración del Módulo Bluetooth.

Este módulo posee la capacidad de trabajar en modo Maestro o Esclavo; tiene una serie de parámetros que se pueden modificar como son, por ejemplo; la velocidad de comunicación con el puerto serie del PC, el nombre del dispositivo, la contraseña del dispositivo, ...

En la figura se puede observar la configuración de todos estos parámetros; en primer lugar, se muestra la conexión realizada con Arduino UNO.

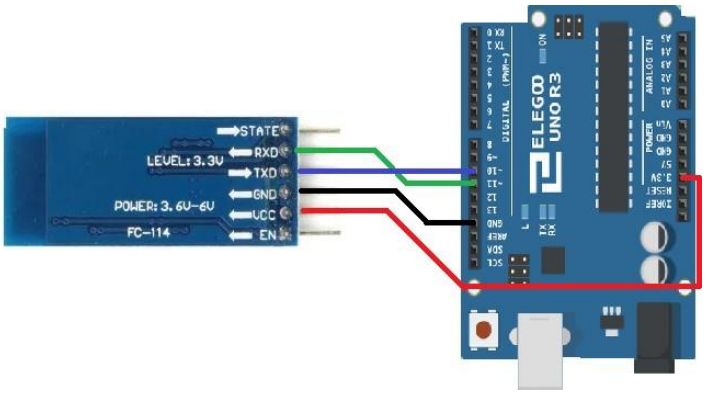

Figura 3.6: Conexionado entre Arduino y HC-05.

Una vez realizada la conexión entre Arduino y HC-05 se deberá conectar Arduino a un PC y cargar el código que permitirá comunicarse con el módulo bluetooth y configurarlo a través de comandos AT. El código es el siguiente:

```
#include <SoftwareSerial.h>      //añadimos la libreria para poder comunicarnos con el serial de IDE.

SoftwareSerial BTT(10,11);      //Definimos los pines por los cuales se realizara la comunicación con
                                //el modulo bluetooth pin 10---->RX y pin 11---->TX

void setup() {
  Serial.begin(9600);            //Iniciamos comunicación serial a velocidad de 9600Bps

  Serial.println("preparado");   // Escribimos en el monitor serial para saber que todo esta correcto.

  BTT.begin(38400);              //Inicializamos la comunicación serie del modulo HC-05 a la velocidad
                                 //de 38400Bps que es la definida por defecto en el modulo
}
void loop() {
 if (BTT.available())           //Mediante la funcion available nos devolvera verdadero si el modulo bluetooth
                                //tiene datos que enviar

 Serial.write(BTT.read());      //Mostramos en el monitor serial los datos que el modulo Bluetooth envia.

 if (Serial.available())        //si escribimos algo en el monitor serial mediante esta funcion la leeremos

 BTT.write(Serial.read());     // Mediante este comando enviamos dicha información al modulo Bluetooth.

}
```

Figura 3.7: Código Arduino de configuración del módulo Bluetooth HC-05.

Los diferentes comandos AT con los que se debe configurar el módulo Bluetooth y sus funciones se muestran en las siguientes tablas.

COMANDO AT	DESCRIPCIÓN
AT+NAME?	Devuelve el nombre del dispositivo
AT+PSWD?	Devuelve la contraseña (PIN)
AT+UART?	Devuelve los parámetros de comunicación
AT+NAME?	Devuelve el rol (0: Esclavo, 1: Maestro)

Tabla 3.2: Comandos AT de consulta

COMANDO AT	DESCRIPCIÓN
AT+NAME=" nombre"	Configura el nombre del dispositivo
AT+PSWD= "1234"	Configura la contraseña del dispositivo
AT+UART="9600,0,0"	Configura la velocidad de comunicación
AT+ROLE="1"	Configura el Rol del dispositivo
AT+ORGL	Restaura los valores de fábrica.
AT+RESET	Sale del modo configuración

Tabla 3.3: Comandos AT de configuración.

3.1.5. Módulo Transceptor Inalámbrico NRF24L01+PA+LNA

Si bien antes se mencionaba la importancia de una comunicación inalámbrica entre el Arduino "Master" y la app, la cual no necesitará conexión a internet, en este apartado se destaca la importancia de una comunicación con las mismas características, pero con una dificultad añadida, la distancia. Una pista de atletismo reglamentaria tiene en total un recorrido de 400 m, divididos en 4 sectores, dos rectas y dos semicírculos, tal como se muestra a continuación:

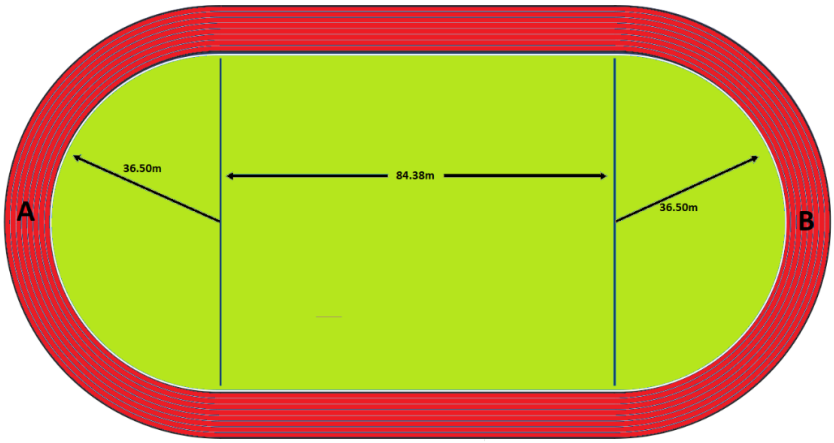

Figura 3.8: Dimensiones de una pista de atletismo.

Las dos curvas de la pista de atletismo tienen un radio de 36.50 m, más 0.2 m si se trata de una pista sin bordillos, por lo que para calcular la longitud de la curva debemos aplicar la siguiente fórmula:

$$\boldsymbol{Longitud\ de\ la\ curva = (36.50 + 0.2) * \pi = 115.29m} \quad (3.1)$$

Al tener dos curvas simétricas y conociendo la medida de las dos rectas obtenemos la medida mencionada:

$$Longitud\ total = 115.29 * 2 + 84.389 * 2 = 400m \quad (3.2)$$

Por último, se debe calcular cual será la distancia máxima a la que se pueden llegar a colocar las puertas, en la Figura 3.8 se ven dibujados los puntos A y B representando la máxima distancia a la que se podrán colocar las puertas, la cual es la siguiente:

$$Distancia\ maxima = (36.50 * 2) + 84.389 = 157.389m \quad (3.2)$$

La distancia máxima a la que deberán comunicarse las puertas es de 157.389 m lo que hace descartar la gran mayoría de tecnologías de comunicación inalámbrica como el bluetooth o wifi. Por lo que se ha optado por una comunicación por Radiofrecuencia la cual tiene un rango de alcance de 10 m a 1000 m, en condiciones óptimas.

De los diferentes módulos de comunicación por radiofrecuencia que existen para Arduino, el más usado por su tamaño y precio es el módulo NRF24L01 del cual hay dos variantes: la versión simple, la cual lleva la antena integrada dentro de la misma placa teniendo un rango máximo de alcance de 30 m, por lo cual es insuficiente, y la variante NRF24L01+PA+LA, la cual incorpora una antena externa que permite aumentar su alcance hasta los 1000 m, siendo más que suficiente para cubrir la distancia deseada.

El NRF24L01 en su versión de largo alcance(+PA+LNA) es el encargado de la comunicación entre las puertas" Esclavo" y la puerta" Master", este módulo es un transceptor (Transmisor y receptor) inalámbrico que trabaja a una frecuencia de 2.4-2.5 GHz (Banda ISM) con una modulación GSFK. Su velocidad de transmisión es configurable (250 kbps,1Mbps y 2Mbps) permitiendo una conexión simultánea de hasta 6 módulos(multiceiver). No es capaz de enviar y recibir a la vez por lo que lo convierte en una comunicación muy robusta, ésta se establece con Arduino mediante interfaz SPI.

La elección de este modelo (NRF24L01+PA+LNA) frente al modelo NRF24L01 es principalmente su chip, que a diferencia de la versión simple incorpora **PA** (Power amplifier), que amplifica la señal que transmite el módulo y **LNA** que tiene como función amplificarla hasta un nivel óptimo la señal recibida extremadamente débil y pequeña que se recibe por la antena hasta un nivel óptimo. Generalmente recibe una señal de µvoltios y se amplifica hasta un rango de (0.5-1V). Esto otorga un alcance de 1000 m en espacios abiertos y sin obstáculos.

Figura 3.9: Modulo NRF24L01+PA+LNA.

Algunas de sus características a destacar son las siguientes:

- Chip RFX2401C.
- Velocidad de hasta 2Mps.
- Rango de hasta 1000m.
- Voltaje de trabajo de 3.3V
- Consumo: Transmisión: 11.3mA, recepción: 13.5mA en 2Mbps.

El dispositivo cuenta con 8 pines de los que se hará uso únicamente de 6, los cuales son:

- GND o pin de tierra.
- **VCC:** Pin de alimentación el cual tolera un voltaje de 1.9 a 3.3V.

- **CE (Chip Enable):** Pin activo a nivel alto, cuando este pin se selecciona el módulo comen- zara a transmitir o recibir.
- **CSN (Chip Select Not):** Pin activo a nivel bajo, cuando se activa el pin el nRF24L01 comienza a escuchar datos en su puerto SPI.
- **SCK (Serial Clock:** Recibe pulsos de reloj proporcionados por el módulo Maestro a través del bus SPI.
- **MOSI ((Master Out Slave In):** Entrada de datos SPI al NRF24L01.
- **MISO (Master In Slave Out)** Salida de datos SPI al NRF24L01.

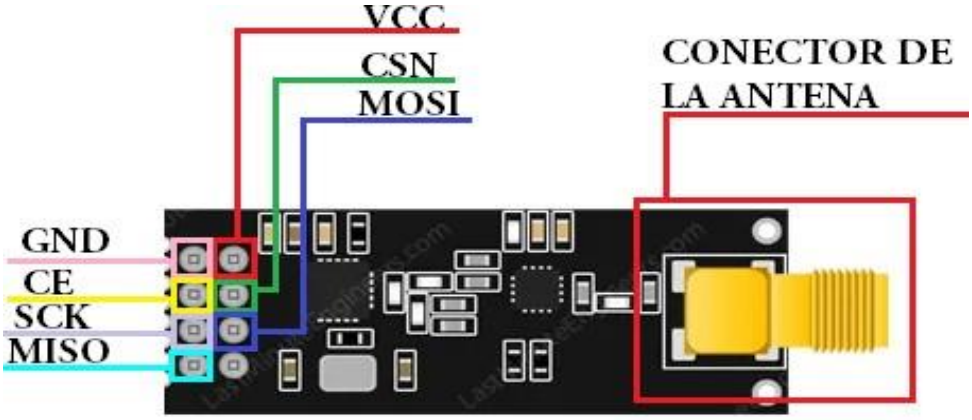

Figura 3.10: PinOut módulo NRF24L01+PA+LNA.

3.1.6. Multiceiver Network.

Multiceiver es una abreviatura de transmisiones múltiples de un solo receptor. Como se ha comentado antes cada Maestro es capaz de comunicarse con 6 esclavos lo que significa que cada canal de RF se divide en 6 canales de datos paralelos (Data Pines) por los cuales se transmite y recibe, cada canal dispone de su propia dirección de canal de datos(configurable).

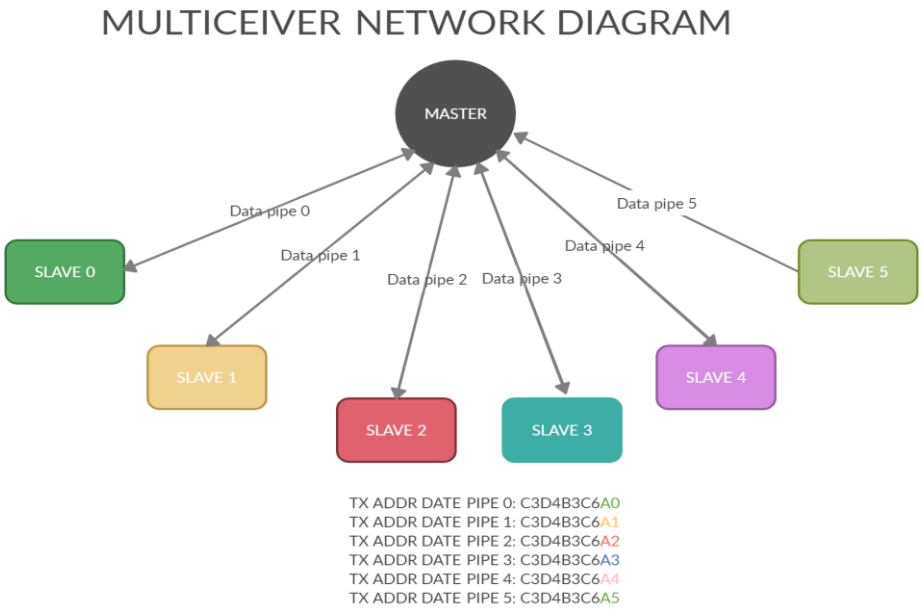

Figura 3.11: Diagrama de la transmisión Multiceiver

Como se aprecia en el diagrama las direcciones de cada nodo son iguales a excepción de los dos últimos bytes, los cuales son diferentes para cada nodo, esto permite al nodo Master saber de dónde procede la información recibida y a donde la envía.

3.2. Sensor Fotoeléctrico XUK9APANM12

El sensor fotoeléctrico reflexivo es el encargado de detectar el paso del corredor al cruzar la puerta cortando el haz de luz de la fotocélula. Se ha escogido una fotocélula de Reflexión, este tipo de sensores contienen tanto emisor como receptor en el mismo plano del encapsulado. El funcionamiento de este tipo de sensores es el siguiente, el emisor emite un haz de luz el cual se refleja en el catadióptrico(reflector) y es detectado por el receptor, cuando el haz de luz es interrumpido por el objeto a detectar, el receptor deja de recibir el haz de luz anulando así la señal de salida del sensor.

El modelo que se ha seleccionado es el **xuk9apanm12** del fabricante Telemecanique, se ha escogido este modelo principalmente por su distancia de trabajo la cual es de hasta 8m, lo que permitiría poder abarcar más de 3 carriles de la pista de atletismo.

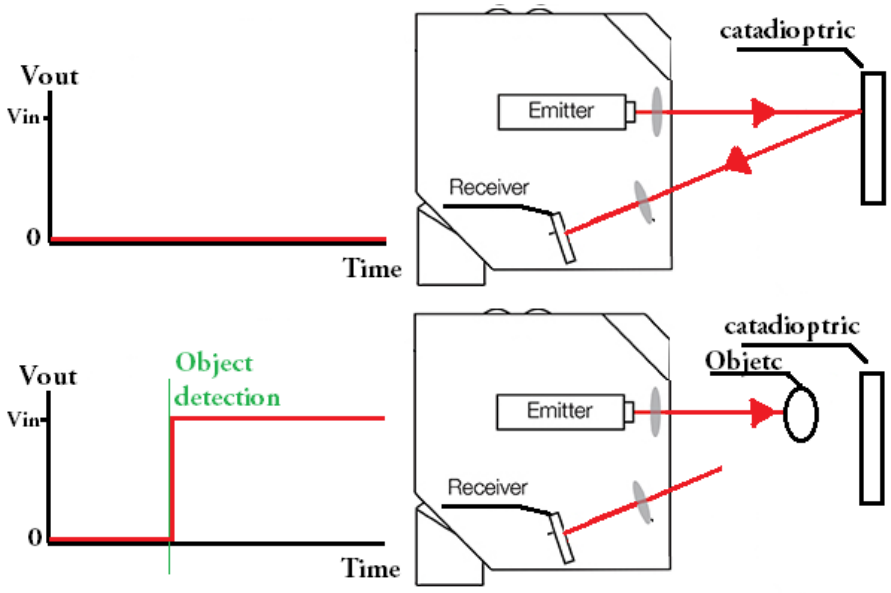

Figura 3.12: Funcionamiento de la fotocélula reflexiva.

A continuación, se recogen algunas de las características del sensor:

- **Tensión nominal de alimentación:** 12-24V.
- **Consumo de corriente:** 35mA sin carga.
- **Frecuencia de conmutación:**250Hz.
- **Maximo delay First Up:**15ms.
- **Tipo de salida digital:** PNP.

3.3. LCD I2C.

La pantalla LCD I2C es una modificación de la clásica pantalla LCD de 16 pines a la cual se le instala un controlador I2c el cual permite controlar la pantalla desde Arduino a través del bus I2C, este usa únicamente 2 cables.

El conexionado con Arduino es simple, como se ha comentado usa únicamente dos cables para la comunicación, un tercero para GND y un cuarto para VCC**.**

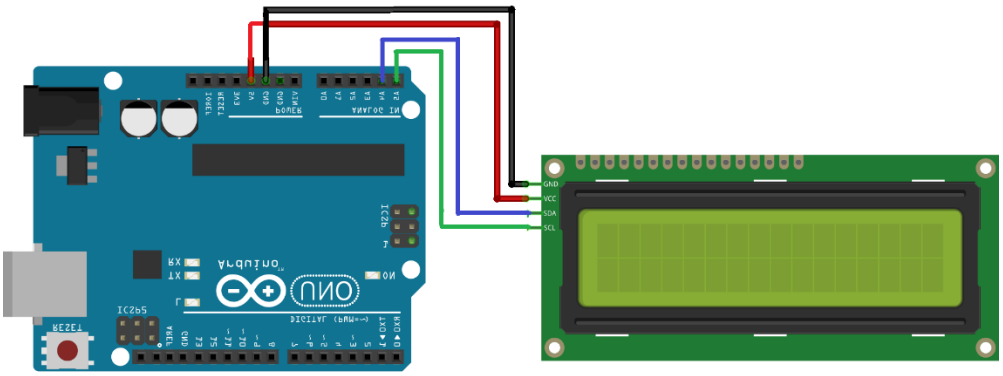

Figura 3.13: Conexión LCD I2C.

3.4. Subsistema de Alimentación del equipo de medida.

La necesidad de que el sistema sea inalámbrico, de bajo coste, y con un tamaño lo más reducido posible, hace indispensable el uso de una batería como fuente de alimentación del equipo. Hay que tener en cuenta que los valores de tensión requeridos por las fotocélulas (12V) y por los Arduino (5V). Los 5V se podrán obtener a partir de los 12V, por lo que este será el valor de partida a la hora de elegir la batería.

Existen varios tipos de batería las cuales podrían ser aptas para este proyecto y con una infinidad de características según voltaje, amperaje hora, tamaño y tecnología.

Los tipos de batería más conocidos y sus características son:

- **Ion de Litio:** Estas baterías son muy interesantes debido a su alta densidad energética en poco espacio, teniendo un peso muy liviano. Poseen un consumo energético muy bajo gracias al litio el cual reduce las pérdidas y tienen un 87 % de rendimiento total. Otro aspecto importante es su larga vida útil admitiendo hasta un rango de 3000-3500 ciclos de carga.

- **AGM:** Tienen una resistencia interna muy baja y una alta resistencia a numerosos ciclos de carga (500 aproximadamente, con una descarga de hasta el 80 %), su tiempo de carga es muy reducido hasta 5 veces menor que el de las baterías de gel.

Se ha escogido una batería AGM de 12V y 2.9Ah, la cual se encarga de proveer el sensor con 12V y al Arduino con 5V necesarios para su correcto funcionamiento. En concreto se ha escogido una batería de la marca ENERGIVM debido a sus características y el precio reducido de esta, por otro lado, el inconveniente principal de esta batería es su abultado tamaño aspecto el cual se mejoraría con baterías de Ion de Litio, pero se descarta por su elevado precio.

- **Voltaje:**12V.
- **Capacidad:** 2.9A.
- **Peso:**1.050gr Aprox.
- **Medidas:**79x56x99mm

A continuación, se muestra la curva de descarga a una temperatura de 25ºC obtenida del datasheet del producto:

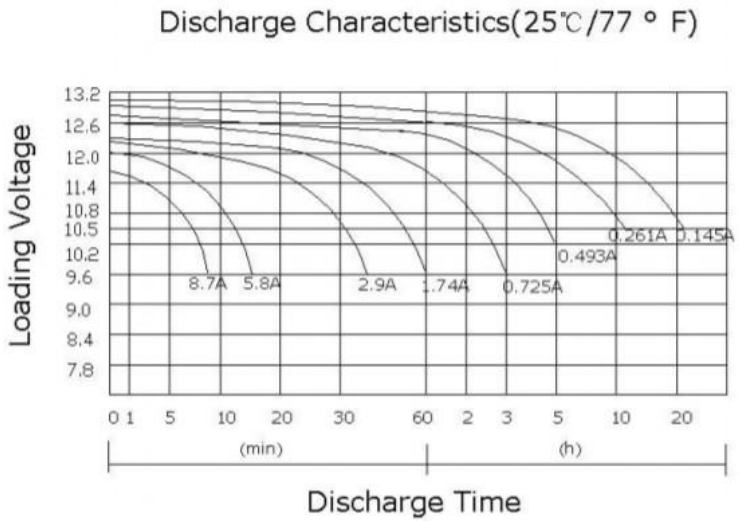

Figura 3.14: curva de descarga de la batería.

3.4.1. Adaptación del Voltaje de Entrada.

Tanto Arduino NANO como Arduino UNO trabajan con una tensión de entrada de 5V lo cual es un voltaje mucho menor que el suministrado por la batería, ante el problema de la adaptación del voltaje de la batería al Arduino se ha decidido buscar un regulador de corriente el cual reducirá la tensión de 12V a 5V para poder alimentar correctamente el Arduino.

Existen diferentes tipos de componentes con los que regular o disminuir un voltaje de forma continua, los más usados son los reguladores lineales y los convertidores buck (del inglés buck converter). Las diferencias más notables entre estos dos componentes son el precio, siendo el convertidor más caro y además permite regular el voltaje de salida haciéndolo más versátil.

Para la elección del componente a usar, se ha hecho un estudio teórico/práctico de los dos tipos, conectando ambos a la misma fuente de entrada (batería de 12V) y una carga a la salida (Led) para así comprobar la potencia consumida y las temperaturas alcanzadas en cada uno, los cálculos obtenidos se muestran a continuación.

Figura 3.15: L7805CV y HW-404.

El consumo de corriente medido a la entrada del regulador linear de voltaje L7805CV con la carga conectada a su salida fue de 0.9A y conociendo su tensión de entrada se puede obtener la potencia consumida:

$$P = I * V = 0.9 * 12.55 = 11.29\text{W} \quad \textbf{(3.4)}$$

Por otro lado, la temperatura de trabajo medida es de 101ºC una temperatura bastante elevada, la cual provoca interrupciones del funcionamiento de dicho regulador, este exceso de temperatura es debido a que el regulador lineal para disipar la diferencia de voltaje entre la entrada y la salida, de 12V a 5V, lo hace en forma térmica, lo que incrementa en exceso la temperatura de trabajo.

Los resultados obtenidos para el buck converter HW-411 han sido mejores. Se ha obtenido una intensidad de entrada de 0.4A, menos de la mitad que en el caso anterior, lo que reduce a más de la mitad la potencia consumida:

$$P = I * V = 0.4 * 12.55 = 5.02\text{W} \quad \textbf{(3.5)}$$

Al disminuir la potencia consumida, se reduce el calor a disipar y con ello la temperatura alcanzada. Se ha obtenido una temperatura de trabajo de 40ºC la cual entra dentro del rango de trabajo del componente y no afecta en absoluto a su funcionamiento.

A la vista de los resultados obtenido se ha elegido es el buck converter HW-411(LM2596) el cual brinda mejores prestaciones, como contra siendo el precio el doble que el del regulador lineal.

3.4.2. LM2596 DC-DC.

La principal función de este convertidor de voltaje es la de entregar un voltaje de salida constante e inferior al voltaje de entrada frente a variaciones en el voltaje de entrada. La gran ventaja de este tipo de reguladores frente a los convencionales son los altos niveles de eficiencia energética que poseen.

Figura 3.16: Regulador de voltaje LM2596.

Su funcionamiento se basa en la conversión de voltajes mediante el almacenamiento periódico de energía de la entrada y su posterior liberación a la salida de una forma controlada para obtener el voltaje deseado.

A continuación, se muestra el esquemático de dicho regulador obtenido del datasheet.

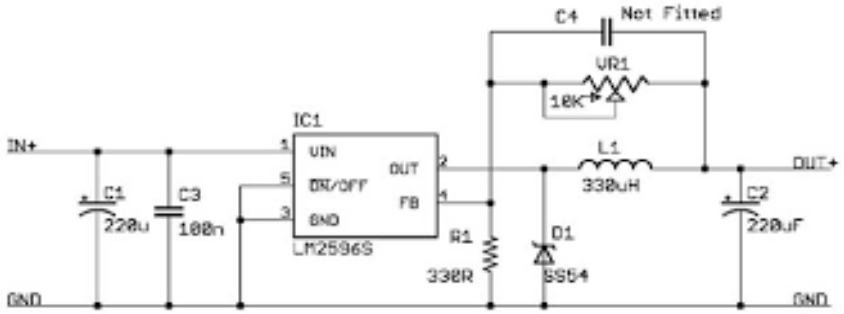

Figura 3.17: Esquema eléctrico LM2596.

Por último, se muestran las principales características del módulo:

- **Tipo de regulador:** Step Down
- **Voltaje de entrada:** 4-40Vdc.
- **Voltaje de salida:** 1.23-35Vdc
- **Temperatura:** -40ºC hasta 85ºC.

3.4.3. Gestión del encendido y apagado del sensor.

Uno de los objetivos marcados al inicio del proyecto es de ahorrar energía, y alargarla duración de la carga de la batería, para ello se expuso la necesidad de gestionar el encendido y apagado de los sensores. De forma que cuando el sistema no estuviese midiendo los tiempos los sensores deberían estar apagados, una vez comenzase la carrera o la calibración se deberían encender.

Para dotar al sistema de esta característica es necesario un switch controlado por una de las salidas digitales de la placa Arduino, la elección de este componente es sencilla, se necesita un relé que soporte una tensión de entrada de hasta 12Vdc.

El modelo más usado para conectar en Arduino, tanto por su simplicidad como por su precio es el relé SRD-05VDC-SL-C; el cual soporta tensiones de hasta 30Vdc. Existen dos montajes de este relé, uno de los montajes viene instalado sobre un PCB con diferentes elementos de protección y el otro es una versión más simple que consiste únicamente en el relé. Para este proyecto se ha elegido la versión simple del relé.

3.4.3.1. Relé SRD-05VDC-SL-C.

Un relé es un dispositivo electromagnético el cual funciona como un interruptor con la particularidad de que es controlado por un circuito eléctrico en el que, a través de una bobina a la cual se le aplica corriente y atrae un electroimán, de forma que acciona un juego de contactos que permite abrir y cerrar el interruptor.

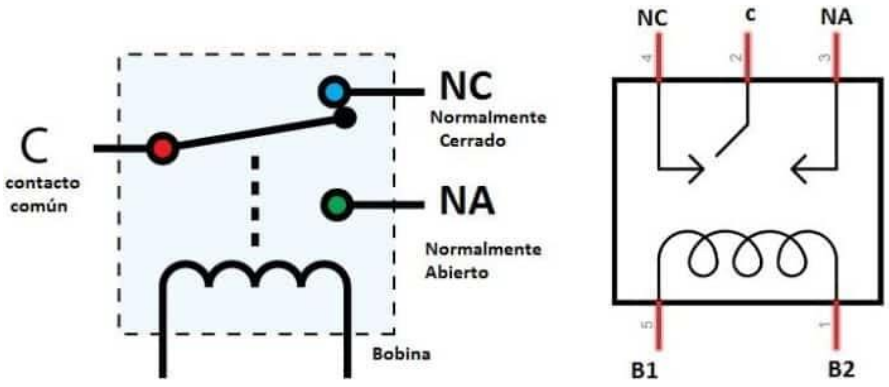

Figura 3.18: Esquema eléctrico de un relé SRD-05VDC-SL-C.

En este proyecto el encendido y apagado del sensor se controlará mediante Arduino. Cuando se desee encender el sensor, Arduino alimentará el relé activándolo y dejando pasar la corriente proveniente de la batería hasta el sensor, de este modo se conseguirá un importante ahorro de energía al mantener el sensor encendido solo cuando sea necesario. Para la protección de las corrientes inversas del relé se usará un diodo 1N4004 y para el acondicionamiento de la señal proveniente de Arduino un transistor 2N2222A junto a una resistencia de 10kΩ conectada a su base, el circuito resultante se muestra a continuación.

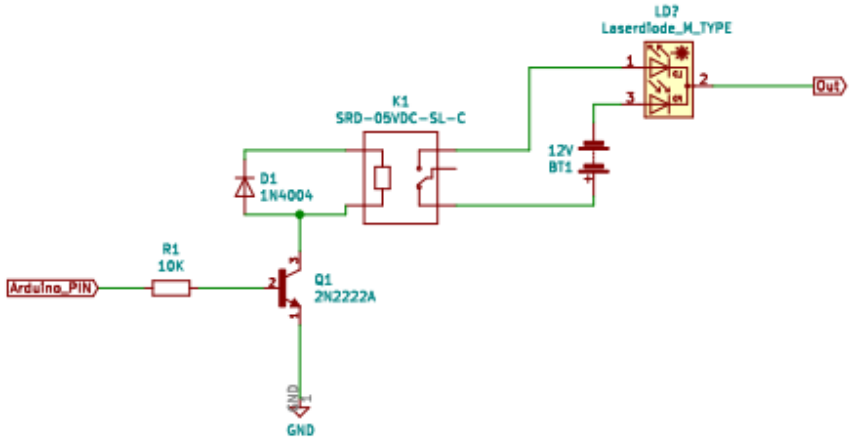

Figura 3.19: Esquema de montaje del relé SRD-05VDC-SL-C.

3.4.4. Acondicionador de la señal de salida del sensor para su entrada al Arduino.

Como se ha explicado con anterioridad, cuando se interrumpe el haz, el sensor desvía el voltaje de entrada y lo muestra por la salida, es decir;

$$V_{out} = V_{in} \tag{3.6}$$

Las entradas analógicas de Arduino solo admiten valores entre 0 y 5V, voltajes superiores a estos dañarían la placa. El valor que se obtiene a la salida del sensor es de uno 12,48V, más del doble del voltaje admitido por Arduino, por ello es necesario acondicionar la señal de salida del sensor y reducirla a valores inferiores a 5V.

Para reducir la tensión se utilizará un divisor de tensiones. Para el diseño del divisor se considerará un voltaje de entrada de hasta 14V.

Por lo que para un voltaje de entrada de 14V y considerando el valor de una de las resistencias del divisor, por ejemplo, R1 igual a 100k. Mediante la siguiente ecuación:

$$V_{out} = \frac{R2}{R2+100K} * 14 \quad \textbf{(3.7)}$$

Sustituyendo Vout por el valor máximo al que opera Arduino (5V) se obtiene el valor de la resistencia R2 de 55K. Para mayor seguridad se ha simulado dicha configuración, obteniendo los resultados esperados. El Arduino se comporta como una impedancia de entrada infinita y no carga al divisor de tensión.

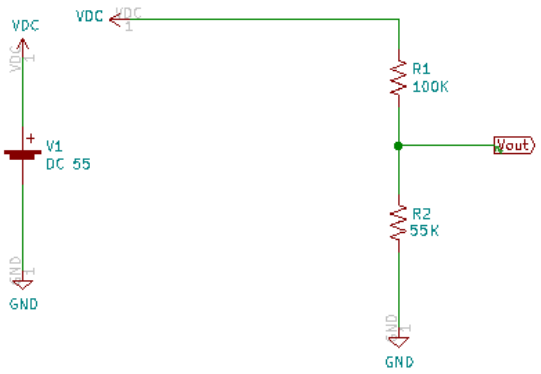

Figura 3.20: Esquema de montaje del divisor de tensión.

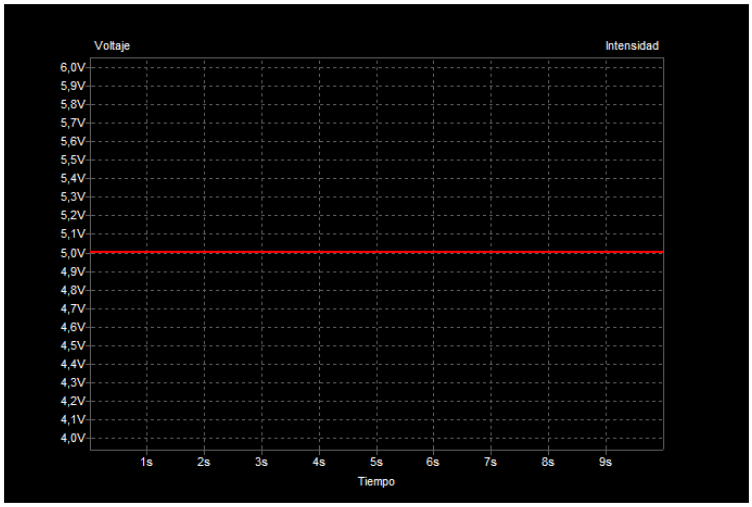

Figura 3.21. Vout del divisor de tensión.

3.4.5. Conexionado de las puertas Master y Esclavo.

A continuación, se muestran los esquemáticos tanto de la puerta "Master" como de la puerta "Esclavo". En la imagen siguiente se observa el esquemático de la puerta "Master" compuesta por un Arduino UNO como cerebro del dispositivo, al cual se le ha conectado un módulo bluetooth HC-05, que será el encargado de comunicar el Arduino con la APP, este se alimenta mediante la salida de 3.3V de Arduino. Los puertos de comunicación RX y TX se conectan a los pines digitales D2 y D3 los cuales se han programado previamente para dicha configuración.

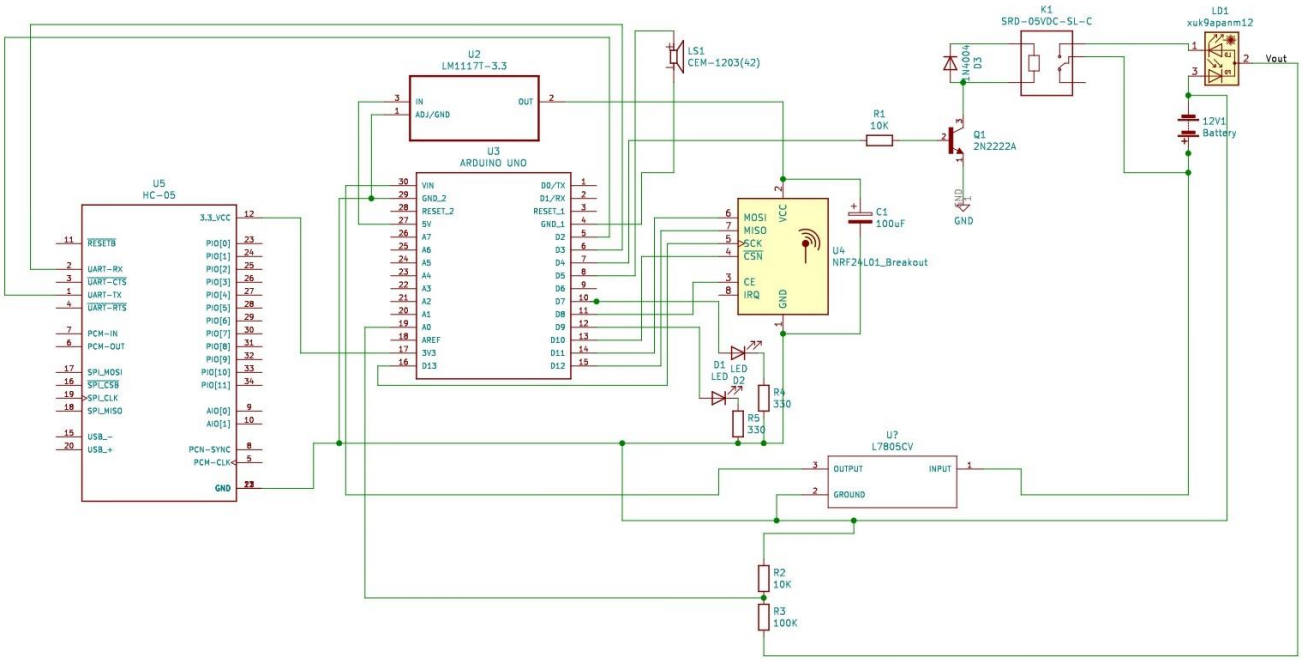

Figura 3.22: Esquemático de la configuración de la puerta Master sin LCD.

A la derecha del Arduino se puede ver el módulo de Radiofrecuencia NRF24l01 encargado de comunicar la puerta "Master" con la puerta "Esclavo". Para alimentar este módulo se ha hecho uso de un regulador de corriente LM1117T para convertir los 5V entregados por la salida de Arduino en 3.3V, tensión de trabajo del módulo de Radiofrecuencia. Esta configuración es necesaria debido a que el módulo nrf24l01 consume 150mA, intensidad que el pin de alimentación de 3.3V de Arduino no es capaz de suministrar.

En la parte inferior del esquemático se observa el módulo HW-411, encargado de regular los 12V que suministra la batería y acondicionarlo a 5V para alimentar la placa Arduino mediante el PIN Vin.

Las resistencias R2 y R3 forman un divisor de tensión conectado al pin

analógicos A0, esta configuración se ha realizado con el fin de medir el voltaje de salida de la fotocélula, mediante este sistema se detectará cuando la puerta ha sido cruzada.

Por último, en la parte superior izquierda del esquemático se encuentra el relé srd-05vdc-sl-c junto a su configuración de protección, el cual está conectado al pin digital D4.

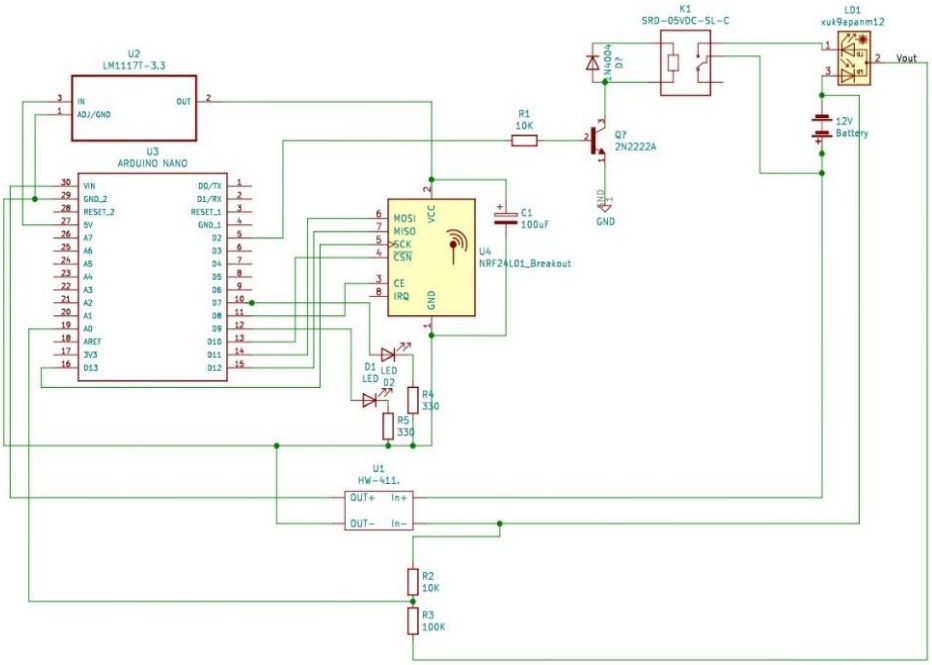

Figura 3.23: Esquemático de la configuración de la puerta Esclavo.

En la anterior figura muestra el conexionado de la puerta "Esclavo" prácticamente idéntica a la anterior, pero en esta configuración se prescinde del módulo bluetooth HC-05 ya que no es necesario. A continuación, se exponen una imagen real de cada módulo.

3.4.6. Consumos y autonomía.

En este apartado se exponen una serie de cálculos, con los cuales se obtendrá una aproximación de la autonomía de la batería tanto del Master como del Esclavo.

En primer lugar, cabe destacar que el consumo de ambas baterías será muy dispar, esto es debido a varios factores como son, Arduino UNO tiene un consumo teórico en vacío de aproximadamente el doble que Arduino NANO. El módulo Master tiene conectado a él un dispositivo Bluetooth el cual incrementa el consumo. Conectando ambos Arduino en vacío a la protoboard y midiendo la intensidad consumida y el voltaje suministrado, se obtiene la potencia consumida.

A continuación, se muestran estas medidas:

	Voltaje(V)	Intensidad(mA)	Potencia(mW)
UNO	5v	45	225
NANO	5v	13	65

Tabla 3.4: Consumo de potencia de los Arduino en vacío.

Para medir el consumo del sensor se debe tener en cuenta sus dos estados, cuando el haz de luz no está interrumpido y cuando sí que lo está. Para ello se han efectuado medidas en ambos casos, dando un consumo claramente superior cuando el haz de luz es interrumpido, así las siguientes medidas son:

	Voltaje(V)	Intensidad(mA)	Potencia(mW)
ON	12.42	18.4	228.528
CUT	12.42	24.9	309.258

Tabla 3.5: Consumo de potencia de la Fotocélula en dos estados.

Se ha Calculado la potencia consumida por los Arduino en vacío, a continuación, se muestran los datos de las placas con sus componentes conectados (bluetooth, RF, relés, reguladores, etc.…)

	Voltaje(V)	Intensidad(mA)	Potencia(mW)
UNO	5v	83.2	416
NANO	5v	47.9	239.2

Tabla 3.6: Consumo de potencia de los Arduino con cargas.

Teniendo en cuenta los datos obtenido del Sensor con el haz de luz cortado y de los Arduino con sus diferentes componentes conectados, se obtiene un consumo total de:

	Potencia(mW)
UNO	416
Fotocélula (cut)	309.258
Total	725.258

Tabla 3.7: Consumo de potencia total del módulo Master.

	Potencia(mW)
NANO	239.2
Fotocélula (cut)	309.258
Total	548.458

Tabla 3.8: Consumo de potencia total del módulo Esclavo.

Por último, según el fabricante esta batería brinda 12V y 2.9Ah en condiciones ideales, lo que da una potencia de 34.8W.

Conociendo la potencia de la batería y la potencia consumida por los módulos se obtiene una estimación de sus autonomías:

$$Autonomia\ Arduino\ UNO = \frac{34.8}{725.258*10^{-3}} = 47{,}98h \quad \textbf{(3.8)}$$

$$Autonomia\ Arduino\ NANO = \frac{34.8}{548.258*10^{-3}} = 63.47h \quad \textbf{(3.5)}$$

Teniendo en cuenta que nuestro sensor necesita una tensión mínima de 8V para funcionar correctamente esta autonomía se reduce 2/3 aproximadamente, lo cual deja una autonomía aproximada para un correcto funcionamiento de:

- Arduino UNO 16h
- Arduino NANO 21h

Para analizar con un poco más de profundidad la descarga de la batería en condiciones reales, se tuvo el módulo Master conectado durante 16h ininterrumpidamente, y se fue midiendo la tensión cada un intervalo de tiempo de 10minutos a través de la Clase volt y graficando las medidas mediante la herramienta Serial:

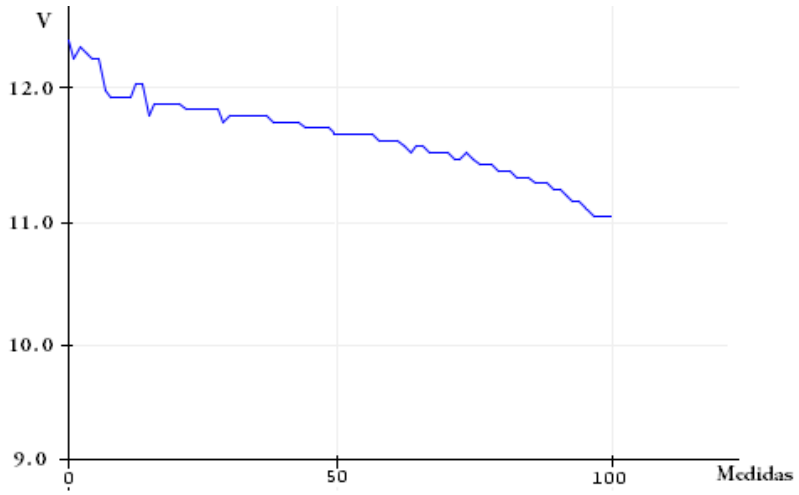

Figura 3.24: Grafica de la descarga de la batería durante 16h.

4. Diseño del Sistema Software.

La parte del diseño de Software de este proyecto consta de dos módulos: la programación de la puerta Master y la programación de la puerta Esclavo. Estos dos bloques se han de comunicar entre si mediante Radiofrecuencia intercambiando información.

Para entender mejor tanto la distribución de la parte Software como la jerarquía, se muestra un esquema ilustrativo:

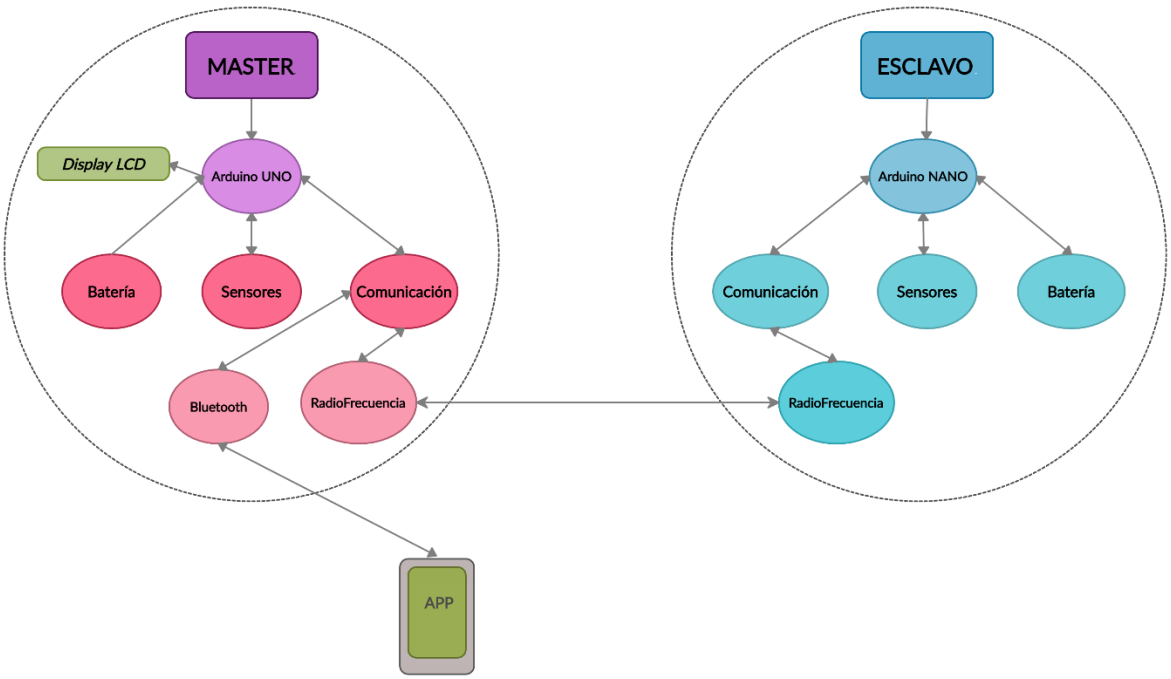

Figura 4.1: Esquema distribución del Software Arduino.

Según se ha comentado en el apartado **3.1.2.4**. la programación del código Arduino se desarrolla en la plataforma IDE propia de Arduino la cual usa un lenguaje propio siendo este una adaptación entre C++ y C.

En primer lugar, se crearon dos Sketch diferentes, uno para el Master y otro para el Esclavo, sobre los cuales se ha ido implementando el código. A continuación, se detalla la implementación realizada.

4.1. Tipos de Variables.

En Arduino como en el resto de lenguajes de programación existen diferentes tipos de variables, dependiendo del tipo dato que se quiera tratar o tamaño, a continuación, se muestran algunas de las más usadas.

- **Int:** Numero entero comprendido entre 32.767 y -32.768 el cual esta codificado en dos octetos (16 bits).

- **Long:** Entero comprendido entre 2.147,483,647 y -2,147,148,648 el cual está codificado en 32 bits (4 bytes/octetos).
- **Float**: Número con decimales comprendido entre 3.4028325E+38 y -3.4028325E+38 almacenado en 4 bytes.
- **Char/String:** cadena de texto que permite almacenar caracteres.
- **Uint8_t:** almacena un valor sin signo de 8 bits(0 a 255).

Clase de Dato.	Memoria que ocupa.	Valores
Int	2 bytes	-32.768 a 32.767
Long	2 bytes	-2,147,148,648 a 2.147,483,647
Float	4 bytes	-3.4028325E+38 a 3.4028325E+38
Char/String	1 byte	-128 a 127
Uint8_t	1byte	0 a 255

Tabla 4.1: Características de cada Variable.

4.2. Código modulo Master.

Como se ha ido explicando a lo largo del presente documento, el módulo master es el encargado de comunicarse con la puerta Esclavo mediante Radiofrecuencia y a su vez con la APP a través de Bluetooth, para ello se debe programar al módulo del código correspondiente para dichas acciones y posteriormente se programó el control de los sensores, leds, etc.

4.2.1 Programación Comunicación Bluetooth.

Primeramente, se desarrolló la parte de código correspondiente al Bluetooth lo cual dotaría al módulo de comunicación directa con la APP mediante el módulo HC-05. Para esto se debe instalar la librería *SoftwareSerial.h* dicha librería permite la comunicación serial mediante otros pines digitales (no solo los pines por defecto 0 y 1), permitiendo así tener diferentes puertos serie simultáneamente con una velocidad máxima de 115200bps.

Como se ve en la primera línea del siguiente código, se ha añadido la librería *SoftwareSerial*.h y en la siguiente línea mediante el comando *SoftwareSerial + (nombre del módulo)* adjudicamos los pines seleccionados para RX (2) y TX (3).

```
#include <SoftwareSerial.h>  // Añadimos la libreria SoftwareSerial
SoftwareSerial miBT(2, 3);   // seleccionamos el pin 2 para RX y el pin 3 para TX del modulo bluetooth
```

Figura 4.2: Librería SoftwareSerial.

A continuación dentro del Void Setup() se ha establecido la velocidad de comunicación serie entre Arduino y el módulo bluetooth, siendo esta de 38400bps.

```
void setup()
{
  miBT.begin(38400);
```

Figura 4.3: Velocidad de comunicación Serie BTT.

Una vez agregada la librería al Sketch y definidos los pines de comunicación así como la velocidad de transmisión serie, se muestra el código en el cual se reciben códigos procedentes de la APP móvil, mediante una estructura *if* dentro del *void loop()* se comprueba si el módulo bluetooth tiene información disponible, si es así mediante la función *readString()* se lee dicha información y se guarda en una variable String, por último se realiza una conversión de dicha variable a una de tipo Int con la función *.toInt()*.

```
if (miBT.available()) {               //Si hay información disponible en el modulo Bluetooth
  receivedData = miBT.readString();   // leemos dicha información y la guardamos en una variable de tipo String.
  timestamp = receivedData.toInt();   // conversión de la cadena receivedData a tipo Int
}
```

Figura 4.4: Recepción de datos proveniente de la APP.

Mediante el código anterior el módulo Master recibe los comandos pertenecientes a las órdenes a ejecutar en nuestro Arduino los cuales son:

- ***"11":*** para comenzar la cuenta atrás de 3 segundo.
- ***"xxxxxxxx"***: se recibe un dato de 8 dígitos el cual es el EPOCH de tiempo.
- ***"22"***: código el cual nos indica que la carrera se ha detenido-
- ***"33"***: al recibir este dato se entra en modo calibración.
- ***"44":*** se sale del modo calibración.

Por último, una vez se obtenido los tiempos conseguidos, se envían dichos datos a la APP mediante la función *println(datoAenviar):*

```
miBT.println(enviar1);   //Enviamos el tiempo de la primera puerta a la APP
```

Figura 4.5: Envió del tiempo obtenido a la APP.

4.2.2 Programación Comunicación RF.

Para la comunicación RF se ha hecho uso de diferentes librerías, estas son *SPI.h* y *RH-NRF24.h.*

La librería *SPI* (Serial Peripheral interface) representa un protocolo de transferencia de datos serie sincronizado, en este tipo de protocolo siempre hay un maestro que controla diversos esclavos.

Por otro lado, la librería *RH-NRF24* es una de las secciones de la librería *RadioHead*, esta sección proporciona funciones para enviar y recibir mensajes (hasta 28 octetos) a cualquier frecuencia admitida por el nRF24L01(2.4-2.5GHz), a una velocidad de datos seleccionada.

```
#include <SPI.h>          //  libreria SPI para comunicacion con el modulo
#include <RH_NRF24.h>     //  seccion NRF24 de la libreria RadioHead
RH_NRF24 nrf24;          // crea objeto con valores por defecto para bus SPI.
```

Figura 4.6: Envió del tiempo obtenido a la APP.

A continuación, dentro del *void setup()* se establecen una serie de comprobaciones para verificar que la conexión de los módulos RF ha sido correcta, estas comprobaciones se muestran de forma visual a través de un Led conectado al pin digital 6 o a través de un mensaje en el puerto serial.

```
  pinMode(6,OUTPUT);

  Serial.begin(9600);   // inicializa monitor serie a la velocidad de  9600 bps
  if (!nrf24.init())    // fallo de inicializacion de modulo
    Serial.println("fallo en la  inicializacion");
     digitalWrite(ledError, HIGH);
  if (!nrf24.setChannel(2)) // fallo al  establecer canal
    Serial.println("fallo al establecer canal");
     digitalWrite(ledError, HIGH);
  if (!nrf24.setRF(RH_NRF24::DataRate250kbps, RH_NRF24::TransmitPower0dBm)) // si falla
    Serial.println("fallo en opciones RF");            // RF muestra texto
     digitalWrite(ledError, HIGH);
```

Figura 4.7: Comprobación de la comunicación RF.

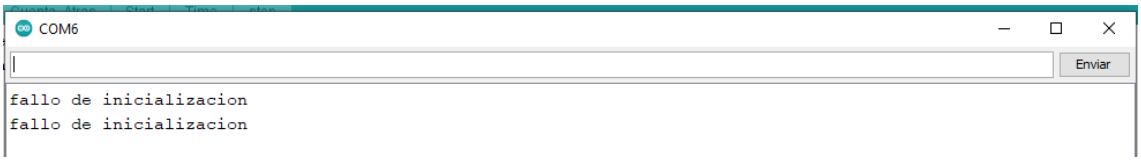

Figura 4.8: Mensaje de fallo de inicialización de la comunicación RF.

Por último, se ha de enviar y recibir datos a través del módulo RF. A la hora de enviar un dato se guarda dicho dato en una variable de tipo uint8_t y mediante la librería nrf24 y la función *send* se envía el dato, una vez enviado mediante la función *waitpacketSent*() el programa se mantiene a la espera de que el envió se realice correctamente.

```
nrf24.send(datal, sizeof(datal));   // envia el texto
nrf24.waitPacketSent();     // espera hasta realizado el envio
```

Figura 4.9: Envió del dato a través de RF.

A la hora de recibir datos, primero debemos crear un buffer de una longitud determinada, en este caso se ha creado un buffer de 32 bytes el cual es suficientemente grande como para almacenar los datos recibidos, posteriormente se obtiene la longitud de dicha cadena.

```
uint8_t buf[RH_NRF24_MAX_MESSAGE_LEN];  // buffer con longitud maxima de 32 bytes
uint8_t len = sizeof(buf);      // obtiene longitud de la cadena
```

Figura 4.10: Creación de un Buffer para almacenar el dato recibido.

Para la recepción de los datos, mediante una concatenación de estructuras if se comprueba si hay información disponible en el módulo y de ser así se guarda en el buffer y se muestra el dato por pantalla.

```
if (nrf24.available())      // si hay informacion disponible
{
  if (nrf24.recv(buf, &len))      // si hay informacion valida en el buffer
  {
    Serial.print("Recibido: ");   // muestra texto
    Serial.println((char*)buf);   // muestra contenido del buffer
    epoch = (char*)buf;
```

Figura 4.11: Recepción de los datos a través del módulo RF.

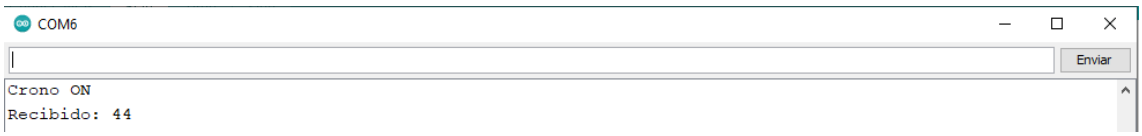

Figura 4.12: Mensaje con el dato recibido mediante RF.

4.2.3 Desarrollo de la parte lógica.

En los dos apartados anteriores se ha explicado la implementación en el código tanto de la comunicación bluetooth como de la comunicación de RF, a continuación, se va a explicar el desarrollo de la parte lógica del módulo Master y su funcionamiento.

El código de este módulo consta de un Sketch principal el cual recibe el nombre ***Master1.0***, este se compone de una parte inicial donde llamar a las librerías o declarar variables.

```
// Programa Master
#include<Wire.h>
#include <LCD.h>
#include<LiquidCrystal_I2C.h>
#include <SPI.h>  // incluye libreria SPI para comunicacion con el modulo
#include <RH_NRF24.h> // incluye la seccion NRF24 de la libreria RadioHead
#include <SoftwareSerial.h>
SoftwareSerial miBT(2, 3);
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7);
RH_NRF24 nrf24;   // crea objeto con valores por defecto para bus SPI y pin digital numero 8 para CE

unsigned long tiempo;
String receivedData = "";
long timestamp;
long tiempoSumar = 0;
String epoch;             //variable dnde se almacena el EPOCH recibido
int ledError = 6;
int ledCalibracion = 1;
int h, m, s, ms;          //variable para la connversion de milisegundos a la hora de mostar display
int longitudD;            //longitud de la prueba
int longitud;             //longitud del mensaje recibido por bluetooth
```

Figura 4.13: Parte inicial del código del módulo Master.

Por otro lado contiene dos bucles principales, clásicos en Arduino, *void setup()* bucle donde se recoge la información como la inicialización de los elementos, bucle el cual solo se ejecutara una vez, para este proyecto. Dicho bucle recoge la inicialización de los pines de entrada y salida, la velocidad de transmisión entre el módulo bluetooth, la velocidad del puerto serial y las diferentes comprobaciones de la correcta conexión RF.

```
void setup()
{
  lcd.setBacklightPin(3, POSITIVE);
  lcd.setBacklight(HIGH);
  lcd.begin(16, 2);
  lcd.clear();
  pinMode(9,OUTPUT);
  pinMode(5,OUTPUT);
  pinMode(7,INPUT);
  pinMode(6,INPUT);
  pinMode(1,OUTPUT);

  miBT.begin(38400);
  Serial.begin(9600);   // inicializa monitor serie a 9600 bps
  if (!nrf24.init())    // si falla inicializacion de modulo muestra texto
    Serial.println("fallo de inicializacion");
  if (!nrf24.setChannel(2)) // si falla establecer canal muestra texto
    Serial.println("fallo en establecer canal");
  if (!nrf24.setRF(RH_NRF24::DataRate250kbps, RH_NRF24::TransmitPower0dBm)) // si falla opciones
    Serial.println("fallo en opciones RF");            // RF muestra texto
}
```

Figura 4.14: Bucle Void Setup del módulo Master.

Por último, en el sketch principal encontramos el módulo *Void loop*, modulo el cual contiene el código que se va a ejecutar continuamente, en él también se encuentran las llamadas a otras clases.

```
void loop()
{
  if (miBT.available()) {
    receivedData = miBT.readString();
    timestamp = receivedData.toInt();
    //char cadena =receivedData.toCharArray(10);
     longitud=  receivedData.length();

  }
  //--------------------RECIBE MENAJE CUENTA ATRAS ON-----------------------------------
  if (timestamp == 11) {
    PreStart();
  }
  //-----------------------RECIBE EPOCH CRONO ON-------------------------------------
  if (longitud >= 6) {
    Run();

  }
  //----------------------RECIBE MENSAJE DE STOP-------------------------------
  if (timestamp == 22) {
    Stop();
  }
  //-------------------------------RECIBE MENSAJE ENTRAR MODO CALIBRACIÓN-------------------------------
  if (timestamp == 33) {
    Calibracion();

  }
  if (timestamp == 44) {
    StopCalibracion();

  }

if (timestamp >=50 && timestamp<1000) {

          longitudD=timestamp;
          lcd.setCursor(11, 0);
          lcd.print("D" );
          lcd.print(timestamp);
          lcd.print("m");
          Serial.println("comeme los huevos");
          Serial.println(longitudD);
          Serial.println(timestamp);

  }
}
```

Figura 4.15: Bucle void loop del módulo Master.

Como se ve en la imagen anterior el bucle loop está formado por una serie de estructuras *if*, comenzando por un *if* en el cual comprobamos si hay información disponible en el módulo bluetooth, si la hay guardamos dicho dato en una variable de tipo *String* que convertimos a tipo *int* para poder tratar mejor con él a la hora de comparar dicho valor, estos datos serán las ordenes enviadas desde la APP, las cuales se han comentado en el apartado **4.2.1.** Seguidamente hay una serie de estructuras *if* las cuales compararan el valor recibido por el módulo bluetooth y en función de dicho valor llamaran a una clase u otra. En la siguiente imagen se pueden ver las diferentes clases.

Figura 4.16: Clases del módulo Master.

Las diferentes clases que componen el Módulo Master se explica a continuación:

4.2.4 Calibración.

Cuando el módulo Bluetooth recibe un dato con valor “33” el bucle principal llama a la clase “calibración” la cual se muestra a continuación:

```
int Calibracion(){

     Serial.println("Modo alineación ON"); // muestra texto
     digitalWrite(ledCalibracion, HIGH);  /se activa el led para mostrar que esta en modo calibración
    uint8_t data2[] = "33";   // almacena texto a enviar
    nrf24.send(data2, sizeof(data2));   // envia el texto
    nrf24.waitPacketSent();     // espera hasta realizado el envio
    digitalWrite(ledCalibracion, HIGH);

  }
```

Figura 4.17: Clase Calibracion del módulo Master.

Dicha clase muestra por monitor serial que ha entrado en modo calibración y enciende un led mediante el pin digital 1, a continuación, almacena en una variable de tipo UINT8_t el valor “33” y lo envía a través de RF al módulo Esclavo

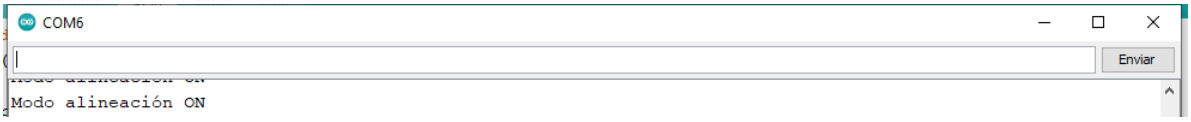

Figura 4.18: Mensaje de que el módulo se encuentra en modo calibración.

4.2.5 StopCalibracion.

En el momento que el módulo HC-05 recibe un dato proveniente de la APP con valor “44”, el bucle principal mediante un *if* llama a la clase *StopCalibracion*.

```
int StopCalibracion(){

     Serial.println("Modo alineación OFF"); // muestra texto
    uint8_t data2[] = "44";   // almacena texto a enviar
    nrf24.send(data2, sizeof(data2));   // envia el texto
    nrf24.waitPacketSent();     // espera hasta realizado el envio
    digitalWrite(ledCalibracion, LOW);

  }
```

Figura 4.19: Clase StopCalibracion del módulo Master.

Una vez se entra en esta clase, se muestra por el puerto serial un mensaje como que la calibración ha concluido y apaga el led indicativo de dicho modo y a continuación manda un dato con valor “44” al módulo Esclavo mediante RF.

Figura 4.20: Mensaje de que el módulo se encuentra fuera del modo calibración.

4.2.6 PreStart

La clase PreStart es la encargada de avisar a los módulos de que al final de la cuenta atrás(3s) comenzara la carrera, por ello esta clase enciende los sensores para que a la hora de comenzar la actividad ya estén listos.

```
int PreStart() {
  Serial.println("Cuenta atras"); // muestra texto
  uint8_t data[] = "00";   // almacena texto a enviar
  nrf24.send(data, sizeof(data));   // envia el texto
  nrf24.waitPacketSent();     // espera hasta realizado el envio
  digitalWrite(5, HIGH);
  delay(1000);
  digitalWrite(5, LOW);
  tiempo = millis();

}
```

Figura 4.21: Clase PreStart del módulo Master.

Cuando el bucle principal llama a esta actividad, esta guarda en una variable de tipo Int el tiempo en milisegundos desde que se encendió la placa mediante la función *millis()*,variable que se usara para calcular el tiempo obtenido en la actividad. Seguidamente se mostrará por pantalla el mensaje de que se encuentra en el modo de “cuenta atrás” y se enviará un dato con valor “00” mediante el módulo de RF encendiendo los sensores.

Figura 4.22: Mensaje de activación de la clase cuenta atrás.

4.2.7 CalTime.

La clase *CalTime()* es la encargada del cálculo transcurrido desde el inicio de la carrera hasta que se corta tanto la primera fotocélula como la segunda.

Para el cálculo de este tiempo se usa la función *millis()* la cual nos devuelve el tiempo desde que se enciende la placa Arduino. Cuando es llamada esta clase se guarda el tiempo de *millis*() en una variable de tipo *unsigned long* llamada *tiempoCruce*, posteriormente en la variable *tiemposumar* se le resta al valor de *tiempoCruce* a la variable *tiempo*, la cual alberga el tiempo transcurrido en el inicio de la carrera, esta resta es el tiempo resultante desde que se inicia la carrera hasta que se cruza alguna de las puertas.

```
String CalTime(){

      unsigned long tiempoCruce = millis();       //guardamos millis() en la variable tiempoCruce(este seral el tiempo
                                                  // desde que se enciende la placa hasta que se corta alguna de las puertas
      tiempoSumar = tiempoCruce - tiempo;         // le restamos al tiempo obtenido al cruzar una puerta el tiempo obtenido al
                                                  //comenzar la carrera, esto sera el tiempo total desde el inicio hasta el corte.
      long tiempoFinal = timestamp + tiempoSumar; // le sumamos al tiempo recibido por la APP el tiempo calculado de la carrera
      String totalString = (String)tiempoFinal;   //lo convertimos a String.
      return totalString;

 }
```

Figura 4.23: Clase CalTime del módulo Master.

4.2.8 Tiempo

La clase *Tiempo* tiene como función el cálculo y posterior adecuamiento de los datos obtenidos en la carrera, el tiempo calculado es en milisegundos por lo que a la hora de mostrarlo es más cómodo convertirlo en minutos, segundo y milisegundos.

Para ello mediante la conversión de segundos a las diferentes medidas deseadas se obtienen las variables de tipo int las cuales se convierte a String para mostrar por el monitor serial

```
String Tiempo(){
      int MostrarTime=CalTime().toInt();          //La variable int MostrarTime pasa a tener el valor del tiempo calculado
                                                  // en la clase CalTime() en milisegundos.

      MostrarTime= MostrarTime-timestamp;         // A este tiempo tenemos que restarle el epoch recibido desde la APP

      unsigned long terminado =MostrarTime%3600000;  // en una variable unsigned log guardamos el valor anterior dividido entre 3600000
                                                     // que son los milisegundos que tiene una hora.

      m=int(terminado/60000);                     //para calcular los minutos dividimos la variable anterior entre 60000 que son los milisegundo
                                                  //que tiene un minuto
      terminado=terminado %60000;
      s=int(terminado/1000);                      //por ultimo dividimos esa variable entre 1000 que son los milisegundos que tiene un segundo
      ms=terminado%1000;
      String mS=(String)m;                        //convertimos todas esas variables en tipo String para mostrar por el terminal serie
      String sS=(String)s;
      String msS=(String)ms;
      String TiempoMostrar=(mS+":"+sS+":"+msS);
      return TiempoMostrar;

 }
```

Figura 4.24: Clase Tiempo del módulo Master

4.2.9 Run.

La clase Run se puede considerar la más importante del módulo, debido a que es la encargada de recoger y calcular los tiempos.

Como en las clases anteriores, primero se muestra por pantalla que se a accedido a la clase *Run* la cual enciende el "crono", a continuación, se envía un dato de valor "20" al módulo esclavo mediante el módulo RF y creamos un buffer donde guardaremos el dato recibido del Esclavo.

Seguidamente la clase se divide en dos partes bien diferenciadas, la parte en la que recibimos una señal de que el sensor ha sido cortado del módulo esclavo y otra en la que se recibe si el sensor(pulsador) del módulo Master ha sido cortado.

En la primera de las partes mediante un if se comprueba si hay información disponible a través del módulo RF, de ser así, se guarda en una variable de tipo *uint8_t*, si este dato tiene como valor "44" significa que el sensor del módulo Esclavo ha sido cortado, por lo que se llama a la función *CalTime*() la cual devuelve el tiempo transcurrido desde que se inició la carrera hasta que se ha cortado el sensor del módulo Esclavo.

Al dato enviado por la clase *CalTime*() se le añade un "00" al principio de la cadena, esto se hace para diferenciar de donde procede el dato. Una vez el dato está compuesto de "01" + "*CalTime*()" se envía mediante el módulo bluetooth a la app.

En la segunda parte de esta clase el proceso es muy similar al anterior, mediante una if comprobamos si el pulsador ha sido accionado, en caso afirmativo se enciende un led a modo de señal mediante el pin digital 9, a continuación se llama a la clase *CalTime*() la cual nos devuelve el tiempo transcurrido desde que se inició la carrera hasta el momento que se ha cortado dicho sensor, a este dato le añadimos un "00" al principio de la cadena para saber la procedencia de dicho dato y mediante el módulo Bluetooth se envía a la APP.

```
int Run() {

  Serial.println("el tiempo de inicio es: ");
  Serial.println(tiempo);
  Serial.println("Crono ON"); // muestra texto
  uint8_t datal[] = "20";   // almacena texto a enviar
  nrf24.send(datal, sizeof(datal));   // envia el texto
  nrf24.waitPacketSent();     // espera hasta realizado el envio
  uint8_t buf[RH_NRF24_MAX_MESSAGE_LEN];  // buffer con longitud maxima de 32 bytes
  uint8_t len = sizeof(buf);      // obtiene longitud de la cadena

  if (digitalRead(5) == HIGH) {
    digitalWrite(9, HIGH);
    CalTime();
    String enviar1 = "00" + CalTime();
    miBT.println(enviar1);
    Serial.print("el tiempo en el modulo master es: ");
    Serial.println(Tiempo());
  } else {
    digitalWrite(9, LOW);

    if (nrf24.available())      // si hay informacion disponible
    {
      if (nrf24.recv(buf, &len))      // si hay informacion valida en el buffer
      {
        Serial.print("Recibido: ");   // muestra texto
        Serial.println((char*)buf);   // muestra contenido del buffer
        epoch = (char*)buf;

        if (epoch == "44") {
          CalTime();
          Serial.println("tiempo transcurrido desd incio  :");
          Serial.println(tiempo);
          Serial.println("resta de los dos tiempor:");
          Serial.println( tiempoSumar);
          Serial.println("el epoch recibido es: " + receivedData);
          String enviar2 = "01" + CalTime();
          miBT.println(enviar2);
          Serial.print("El tiempo en el modulo Esclavo es: ");
          Serial.println(Tiempo());;

        }
      }
    }
  }
}
```

Figura 4.25: Clase Run del módulo Master.

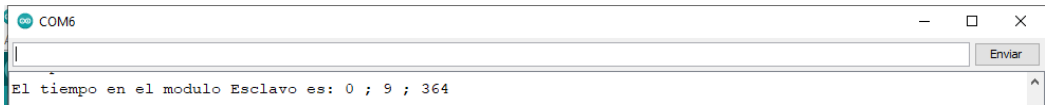

Figura 4.26: Tiempo mostrado por monitor serial al cruzar la puerta Esclavo.

Figura 4.27: Tiempo mostrado por monitor serial al cruzar la puerta Master.

4.2.10 Stop

La última clase, mediante la cual se reinicia las variables y apagan los sensores. Cuando el bucle principal llama a esta clase es debido a que la APP ha enviado mediante el módulo Bluetooth la señal de Stop.

```
int Stop(){

Serial.println("Crono fin"); // muestra texto
    uint8_t data2[] = "11";   // almacena texto a enviar
    nrf24.send(data2, sizeof(data2));   // envia el texto
    nrf24.waitPacketSent();     // espera hasta realizado el envio
    Serial.println("es un 11");
      lcd.clear();


  }
```

Figura 4.28: Clase Stop del módulo Master.

4.3 Código Módulo Esclavo.

En los apartados anteriores se ha explicado la programación del módulo Master, en este apartado se explicarán las diferentes partes del módulo Esclavo el cual no es muy distinto del anterior.

Como se ha mencionado a lo largo de los anteriores apartados, el módulo Esclavo se comunica mediante RF con el módulo Master del cual recibe y le envía información, esta parte de la programación ya ha sido explicada en el apartado anterior. Por otro lado, el módulo ha sido programado para ser capaz de leer los sensores, activar los leds o tratar la información recibida de forma correcta.

4.3.1 Clase Volt.

Es la única clase que contiene el módulo Esclavo, la función de esta es la medición de el voltaje a la salida del sensor, mediante el pin analógico A0 leemos el voltaje procedente del divisor de tensión explicado en el apartado de **Hardware**, una vez obtenido dicho valor lo multiplicamos por el voltaje de referencia el cual lee dicho pin (0-4.95V) y dividimos entre 1024 valores, que es el conversor de 10 bits. Por último, calculamos el divisor de tensión aplicado y obtenemos el valor del voltaje.

```
float voltaje_entrada;
float voltaje_final;
float resistencial = 100000; //Resistencia de 100K
float resistencia2 = 10000; //Resistencia de 10k
int volt(){

 voltaje_entrada = (analogRead(A0) * 4.95) / 1024;  //Lee el voltaje de entrada
  voltaje_final = voltaje_entrada / (resistencia2 / (resistencial + resistencia2));  //Fórmula del divisor resistivo para el voltaje final
  return voltaje_final;


}
```

Figura 4.29: Clase Volt.

4.3.2 Desarrollo de la parte lógica.

El módulo Esclavo está compuesto por un sketch principal llamado Esclavo en el cual como se ha explicado anteriormente se divide en tres partes; una parte inicial donde se instancian las librerías y las diferentes variables que se deseen usar.

```
RH_NRF24 nrf24;              // crea objeto con valores por defecto para bus SPI
int pin_lectura = A0;        //pin por el cual se mide el voltaje de salida del sensor
int led = 9;                  //led que nos indicara si nos encontramos en modo calibración
                              // o si se ha detectado algun corredor.
int relay = 2;                //pin de activación del rele.
String cadena;
boolean modoCarrera = false;
```

Figura 4.30: Parte inicial del Sketch Esclavo

A continuación, un bucle void setup el cual se ejecuta una única vez y en el cual se recoge toda la información de tipos de variables (IMPUT, OUTPUT…), velocidades de transmisión y una serie de comprobaciones.

```
void setup()
{
  pinMode(pin_lectura, INPUT);
  pinMode(ledError, OUTPUT);
  pinMode(led, OUTPUT);
  pinMode(relay, OUTPUT);
  Serial.begin(9600);   // inicializa monitor serie a 9600 bps
  if (!nrf24.init())    // si falla inicializacion de modulo muestra texto
    Serial.println("fallo de inicializacion");
  digitalWrite(ledError, HIGH);
  if (!nrf24.setChannel(2)) // si falla establecer canal muestra texto
    Serial.println("fallo al establecer canal");
  digitalWrite(ledError, HIGH);
  if (!nrf24.setRF(RH_NRF24::DataRate250kbps, RH_NRF24::TransmitPower0dBm)) // si falla opciones
    Serial.println("fallo en las opciones RF");            // RF muestra texto
  digitalWrite(ledError, HIGH);
  Serial.println("Base iniciada correctamente");  // texto para no comenzar con ventana vacia

}
```

Figura 4.31: Void Setup del Sketch Esclavo

La parte principal del código, el Void loop está compuesto por una concatenación de if y un while. Mediante estructura if se comprueba si el dato recibido por RF y almacenado en un buffer corresponde a alguna de las acciones, de ser así entra en dicho bucle y ejecuta las instrucciones que hay dentro.

A continuación, se muestra la estructura if perteneciente a la activación de la cuenta atrás, cuando se recibe mediante RF un dato de valor “00” el programa enciende el led del pin digital 9 para advertir que el módulo está en modo calibración y seguidamente activa el relé que acciona la alimentación del sensor.

```
//------------------------CUENTA ATRAS-----------------------------------
if (cadena == "00") {
  digitalWrite(led, HIGH);
  digitalWrite(relay, HIGH);  //activamos el sensor
}
```

Figura 4.32: Bloque if encargado de la cuenta atrás.

Una vez se acaba la cuenta atrás y se recibe el mensaje de Start el cual tiene un valor de "20" mediante un if asignamos a la variable de tipo booleana el valor de true, de este modo nuestro programa accede al while.

```
//-----------------CRONO ON-------------------------------------------------
if (cadena == "20") {
 modoCarrera = true;
}
```

Figura 4.33: Bloque if encargado activar el crono.

Una vez dentro del bucle while bajo la condición de modoCarrera=true, comprobamos si hay información disponible en el módulo RF, si esto fuera cierto el programa saldrá de dicho bucle, en caso contrario se activa el relé para dar alimentación al sensor y en una estructura if se llama a la clase Volt() la cual mide el voltaje a la salida del sensor, en caso de que este valor fuese mayor a 2, significaría que el haz de luz del sensor está siendo cortado, por lo que se activa el led que indica el corte del sensor y mediante el módulo RF se envía al módulo Maestro el dato "44", en caso contrario el led permanece apagado.

```
//----------------WHILE DE CRONO ON-----------------------------------
while (modoCarrera == true) {
  if (nrf24.available()) {
    modoCarrera = false;
  } else {
    Serial.println(volt());
    digitalWrite(relay, HIGH);
    if (volt() > 2) {     //si el voltaje supera los 2V la puerta ha sido cortada entonces enviamos señal
      digitalWrite(led, HIGH);
      uint8_t data[] = "44";   // almacena texto a enviar
      nrf24.send(data, sizeof(data));   // envia el texto
      nrf24.waitPacketSent();     // espera hasta realizado el envio
    } else {
      digitalWrite(led, LOW);
    }
  }
}
```

Figura 4.34: Bloque while encargado de tratar la información del sensor.

Por último, se muestran los dos bloques if pertenecientes a la activación y desactivación del modo de alineación de los sensores, en caso de recibir el dato "33" se activaría el relé que alimenta el sensor y el led a modo de advertencia de que el módulo se encuentra en calibración.

Si por el contrario el mensaje recibido es "44" se desconectaría tanto el relé como el led indicativo.

```
  //-----------------CALIBRACIÓN SENSORES ON-------------------------------
  if (cadena == "33") {
    digitalWrite(relay, HIGH);
    digitalWrite(ledError,HIGH);
  }
  //-----------------CALIBRACIÓN SENSORES OFF------------------------------
  if (cadena == "44") {
    digitalWrite(relay, LOW);
     digitalWrite(ledError,LOW);
  }
}
```

Figura 4.35: Bloques if encargado de activar y desactivar el modo alineación.

5. Características y Funcionamiento del Equipo.

5.1. Características.

Una vez finalizada la explicación de las diferentes partes que forman este proyecto, se va a hacer un repaso por las características del sistema.

El sistema está formado por dos módulos de medida de tiempo, dotados de comunicación inalámbrica y batería para un funcionamiento plenamente inalámbrico

El módulo principal, también llamado “modulo Master” posee las siguientes características;

- Comunicación inalámbrica con Aplicaciones Android mediante tecnología Bluetooth a través de un módulo HC-05 V2.0.
- Comunicación inalámbrica por Radiofrecuencia con otros módulos de RF a través de un módulo NRF24L01+PA+LA, con un alcance de hasta 1000m.
- Sistema de ahorro de energía permitiendo la activación y desactivación del sensor.
- Autonomía de hasta 28h.
- Fotocélula Reflexiva con un alcance de hasta 9m y delay máximo de 2ms.
- Indicaciones led del estado del módulo.
- Display LCD donde ver el tiempo y velocidad obtenido.

Por otro lado, el “modulo Esclavo” tiene las siguientes características;

- Comunicación inalámbrica por Radiofrecuencia con otros módulos de RF a través de un módulo NRF24L01+PA+LA, con un alcance de hasta 1000m.
- Sistema de ahorro de energía permitiendo la activación y desactivación del sensor.
- Autonomía de hasta 42h.
- Fotocélula Reflexiva con un alcance de hasta 9m y delay máximo de 2ms.
- Indicaciones led del estado del módulo.

5.2 Funcionamiento.

A lo largo del documento se ha explicado los diferentes aspectos de cada parte del proyecto, así como su funcionamiento entre sí. A continuación, se van a detallar las instrucciones para el uso del dispositivo, así como su funcionamiento general.

5.2.1. Instrucciones de uso.

Los diferentes pasos a seguir a la hora de usar el sistema son los siguientes:

Instrucciones de uso sin PC:

1. Encienda el interruptor que alimenta el sistema tanto del módulo master como del módulo esclavo.
2. Coloque los módulos en las distancias deseadas.
3. Seleccione en la APP el modo calibración.
4. Alinee los sensores con los catadióptricos, si los sensores no están bien alineados se encenderá el led de color rojo
5. Salga del modo alineación, vera como los sensores se vuelven a apagar.
6. Seleccione la distancia deseada y presione el botón INFO de la APP.
7. Presione el botón de Start en la APP, vera como los sensores se encienden y escucha un pitido.
8. Colóquese en el punto de salida.
9. Cuando la señal acústica cese, comience la carrera.
10. Una vez acabada la carrera, compruebe en el display o en la APP su tiempo.

Instrucciones de uso con PC:

1. Encienda el interruptor que alimenta el sistema del módulo esclavo.
2. Conecte el modulo Master al PC, cargue el Sketch “Master1.0.ino” y abra el puerto serial.
3. Coloque los módulos en las distancias deseadas.
4. Seleccione en la APP el modo calibración.
5. Alinee los sensores con los catadióptricos, si los sensores no están bien alineados se encenderá el led de color rojo
6. Salga del modo alineación, vera como los sensores se vuelven a apagar.
7. Presione el botón de Start en la APP, vera como los sensores se encienden y escucha un pitido.
8. Colóquese en el punto de salida.
9. Cuando la señal acústica cese, comience la carrera.
10. Una vez acabada la carrera, compruebe en el display, en la APP o en el monitor serial su resultado.

6. *Resultados y Conclusiones.*

En este apartado se recogen los diferentes ensayos y resultados obtenidos a lo largo de la realización del proyecto, así como un pequeño análisis acerca de los logros conseguidos.

6.1. Resultados.

Al igual que se ha estructurado la explicación del Software en capítulos anteriores, se han ido realizando las comprobaciones y obteniendo los resultados.

En primer lugar, se comprobó el correcto funcionamiento del módulo bluetooth, para ello mediante un código de prueba el cual se muestra a continuación:

```
#include <SoftwareSerial.h>

SoftwareSerial miBT(2, 3);  //

char DATO = 0;
int LEDR = 2;
int LEDAZ = 3;

void setup(){
  miBT.begin(38400);
  pinMode(LEDR, OUTPUT);
  pinMode(LEDAZ, OUTPUT);
}

void loop(){
if (miBT.available()){
  DATO = miBT.read();

  if( DATO == '1' )
  digitalWrite(LEDR, HIGH);

  if( DATO == '2' )
  digitalWrite(LEDR, LOW);

  if( DATO == '3' )
  digitalWrite(LEDAZ, HIGH);

  if( DATO == '4' )
  digitalWrite(LEDAZ, LOW);
}
}
```

Figura 6.1: Sketch comprobación modulo Bluetooth.

Y mediante una APP obtenida de la Play Store se comprobó si efectivamente, se establecía una conexión bluetooth correcta y se manejaban los diferentes leds del montaje que tal como se muestra a continuación, donde se aprecia uno de los leds encendidos mediante la APP:

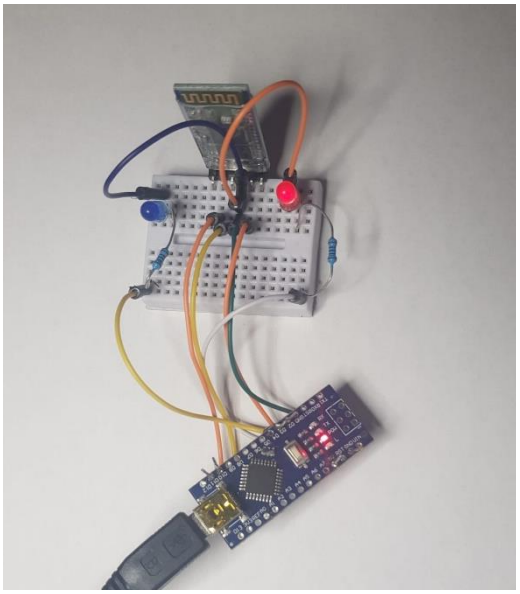

Figura 6.2: Montaje comprobación Bluetooth.

Una vez se comprobó el correcto funcionamiento del módulo Bluetooth se pasó a comprobar la conexión RF, para ello se montaron ambos módulos con sus correspondientes dispositivos RF y se cargaron los códigos que se muestran a continuación:

```
#include <SPI.h>
#include <RH_NRF24.h>

RH_NRF24 nrf24;

void setup()
{
  Serial.begin(9600);   // inicializa monitor serie a 9600 bps
  if (!nrf24.init())    // si falla inicializacion de modulo muestra texto
    Serial.println("fallo de inicializacion");
  if (!nrf24.setChannel(2)) // si falla establecer canal muestra texto
    Serial.println("fallo en establecer canal");
  if (!nrf24.setRF(RH_NRF24::DataRate250kbps, RH_NRF24::TransmitPower0dBm)) // si falla opciones
    Serial.println("fallo en opciones RF");            // RF muestra texto

    Serial.println("Base iniciada correctamente");

void loop()
{
  if (nrf24.available())      /
  {
    uint8_t buf[RH_NRF24_MAX_MESSAGE_LEN];
    uint8_t len = sizeof(buf);
    if (nrf24.recv(buf, &len))
    {
      Serial.print("Recibido mensaje de el esclavo: ");
      Serial.println((char*)buf);

      uint8_t data[] = "Hola desde el modulo Master";
      nrf24.send(data, sizeof(data));
      nrf24.waitPacketSent();
      Serial.println("Respondiendo");
    }
    else
    {
      Serial.println("fallo en recepcion"); // muestra texto
    }
  }
}
```

Figura 6.3: Código Master de comprobación de la comunicación RF.

```
#include <SPI.h>
#include <RH_NRF24.h>

RH_NRF24 nrf24;

void setup()
{
  Serial.begin(9600);
  if (!nrf24.init())
    Serial.println("fallo de inicializacion");
  if (!nrf24.setChannel(2))
    Serial.println("fallo en establecer canal");
  if (!nrf24.setRF(RH_NRF24::DataRate250kbps, RH_NRF24::TransmitPower0dBm))
    Serial.println("fallo en opciones RF");
}

void loop()
{
  Serial.println("Enviando mensaje a la base");
  uint8_t data[] = "Saludos desde el modulo Esclavo";
  nrf24.send(data, sizeof(data));

  nrf24.waitPacketSent();

  uint8_t buf[RH_NRF24_MAX_MESSAGE_LEN];
  uint8_t len = sizeof(buf);

  if (nrf24.available())
  {
    if (nrf24.recv(buf, &len))
    {
      Serial.print("Recibido: ");
      Serial.println((char*)buf);
    }
    else
    {
      Serial.println("fallo en recepcion");
    }
  }
  delay(1000);
}
```

Figura 6.4: Código Esclavo de comprobación de la comunicación RF.

Mostrándose por el monitor serie los mensajes de comprobación de que la comunicación está siendo correcta.

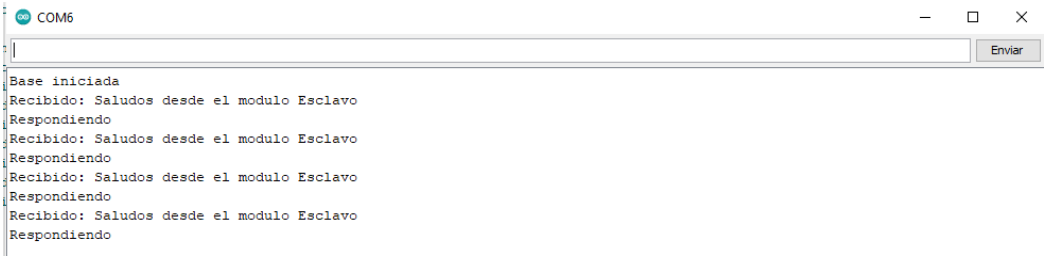

Figura 6.5: Cadena de mensajes de comprobación de la comunicación RF.

Por último, una vez completado el código de ambos módulos, se comprobó el correcto funcionamiento de todas las funciones del sistema; estas comprobaciones se realizaron mediante los diodos leds y las señales acústicas, así como por el display. A continuación, se recogen una serie de imágenes donde se muestra el sistema completo y los resultados obtenidos, tanto en el monitor serial como en los indicadores led.

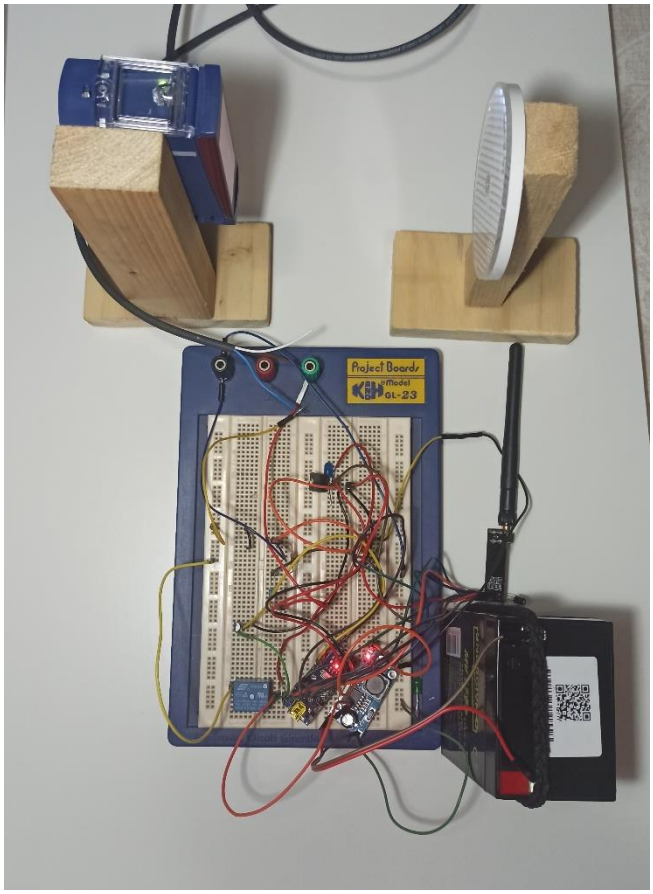

Figura 6.6: Montaje módulo Esclavo.

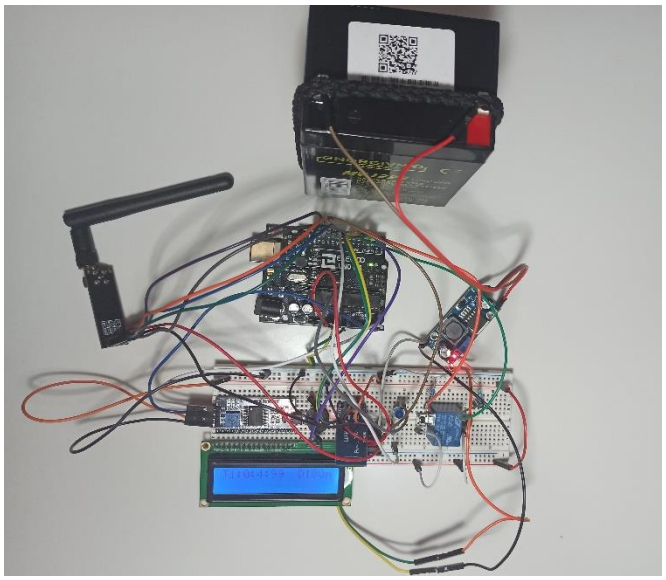

Figura 6.67: Montaje módulo Master.

Figura 6.8: LCD mostrando tiempos, distancia y velocidad media.

Figura 6.9: Monitor Serial mostrando las diferentes medidas obtenidas.

Mediante estas pruebas se comprobó que el sistema funcionaba correctamente, con el único inconveniente del delay provocado por la comunicación bluetooth el cual es aproximadamente de 1,4s un tiempo muy elevado para la medición de tiempos en pruebas de atletismo, por ello se optó por un sistema de tratamiento del tiempo que por el cual el delay no afectara a las medidas del tiempo.

Cuando se recibe el EPOCH con el tiempo proveniente de la APP mediante Bluetooth, este tiempo se guarda en una variable y es el Arduino quien mediante la función *millis()* obtiene el tiempo exacto en el que se ha comenzado la carrera en Arduino, una vez se cruzan las puertas y la carrera finaliza, Arduino muestra ese tiempo por el LCD y por otro lado lo suma a la variable con el EPOCH recibido al comienzo de la prueba y lo envía de nuevo a la APP, con este sistema se evitan los delay superiores a 2ms en la obtención del tiempo de la prueba. Para poder entenderlo con mayor facilidad se muestra un diagrama.

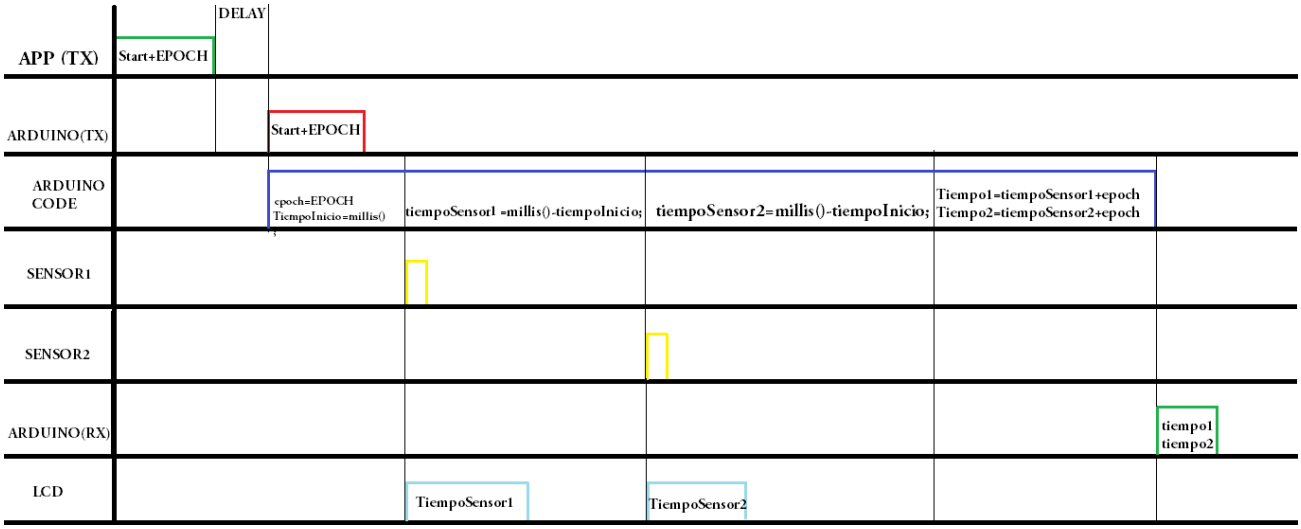

Figura 6.10: diagrama del sistema de eliminación del delay en las medidas obtenidas.

Como se ha comentado, con este sistema no se elimina el delay en la comunicación Bluetooth, lo que se consigue es eliminar ese delay a la hora de tomar las medidas, dando como resultado unas medidas más fiables.

6.2. Conclusiones

Durante la realización del proyecto, se ha diseñado e implementado un Equipo de medida de tiempo para pruebas de atletismo, el cual mediante fotocélulas reflexivas controladas por microprocesadores Arduino detectan cuando el corredor ha cruzado dicho sensor pudiendo así medir los diferentes tiempos y la velocidad media de la carrera. Los diferentes parámetros de la carrera son gestionados mediante una APP externa.

Con este proyecto se buscaba disponer de un sistema de medida con la máxima fiabilidad posible y a bajo coste el cual fuese fácil e intuitivo de manejar.

Se puede decir que se ha podido cumplir con los objetivos propuestos marcados al inicio de este proyecto, habiendo diseñado e implementado un Equipo de medida con un error de medida máximo de 150 ms, totalmente inalámbrico y autónomo, con una autonomía superior a 20h.

A la hora de diferenciarlo de otros equipos comerciales de mayor fiabilidad y un precio 10 veces superior, se le ha dotado de las siguientes características:

- Mediante la gestión del encendido/apagado de los sensores conseguimos una importante disminución del consumo.
- Se ha dotado al equipo de comunicación bluetooth con la que recibir y enviar los diferentes parámetros y resultados de la actividad, lo que permite al equipo no tener que depender de una consola propia externa. Simplemente será necesario el equipo de medida y un terminal Android.
- Se le ha dotado de un sistema RF con el que poder separar los módulos una distancia de hasta 1000m sin perder la conexión.
- Al haber sido programado en Arduino, el proyecto puede ser mejorado y actualizado con facilidad.

El diseño realizado presenta algunas limitaciones frente a otros equipos comerciales, como son:

- Tamaño superior al deseado.
- Delay en la conexión bluetooth.
- Necesidad de reprogramar si se desea conectar otros módulos adicionales.

7. *Líneas Futuras*

A la hora de analizar el sistema saltan a la vista una serie de mejoras que con un presupuesto mayor hubiesen sido posibles como son:

- una reducción del tamaño, haciendo uso de batería de litio, lo que disminuiría tanto el peso como el tamaño, una reducción del tamaño al poder diseñar y fabricar nuestro propio microcontrolador y la PCB correspondiente en la que incluir todos los componentes usados.
- Incluir como medio de comunicación los pulsómetros, los cuales mediante tecnológica bluetooth enviarán información al Arduino.

8. *Presupuesto.*

En este apartado se muestra un presupuesto detallado del coste material del proyecto, así como los diferentes enlaces de compra a cada componente.

COMPONENTE	MODELO	FABRICANTE	UNIDADES	PRECIO UND SIN IVA	PRECIO UND CON IVA	PRECIO TOTAL CON IVA
Microcontrolador	Arduino UNO R3	ELEGOO	1	8,26 €	9,99 €	9,99 €
Modulo Bluetooth	HC-05	OEM	1	8,26 €	9,99 €	9,99 €
Modulo RF	NRF24L01+PA+LA	Nordic Semiconductor	1	5,45 €	6,95 €	6,95 €
regulador de voltaje 3.3V	LM1117T-3.3/NOPB	Texas Instrument	1	1,31 €	1,59 €	1,59 €
Diodo LED	C503B-RCS-CW0Z0AA1	CREE	2	0,14 €	0,17 €	0,35 €
Resistencia 10K	10k 1/2w 5%	Generico	2	0,06 €	0,07 €	0,14 €
Resistencia 100K	100k 1/4w 5%	Generico	1	0,05 €	0,06 €	0,06 €
Resistencia 330	30 Ohm 1w 5%	Generico	2	0,11 €	0,13 €	0,26 €
Fotocélula Reflexiva	XUK9APANM12	Telemechanique	1	62,60 €	75,75 €	75,75 €
Alimentador Regulable	LM2596 HW-411	Generico	1	2,00 €	2,42 €	2,42 €
Relé	SRD-05VDC-SL-C	ELEC&Lifes	1	0,54 €	0,65 €	0,65 €
Transistor	PN2222ATA	ON SEMICONDUCTOR	1	0,13 €	0,16 €	0,16 €
Diodo rectificador	1N4004-E3/53	VISHAY	1	0,35 €	0,42 €	0,42 €
Pantalla LCD	IIC / I2C 1602	Geekcreit	1	2,09 €	2,53 €	2,53 €
protoboard	PSG-BB-400	PRO-SIGNAL	1	2,22 €	2,69 €	2,69 €
batería 12V	AGM 12V-2.9AH MV1229	ENERGIvM	1	10,16 €	12,29 €	12,29 €
			TOTAL:	103,73 €	125,86 €	126,24 €

Tabla 7.1: Presupuesto módulo Master.

COMPONENTE	MODELO	FABRICANTE	UNIDADES	PRECIO UND SIN IVA	PRECIO UND CON IVA	PRECIO TOTAL CON IVA
Microcontrolador	Arduino NANO V3	ELEGOO	1	3,85 €	4,66 €	4,66 €
regulador de voltaje 3.3V	LM1117T-3.3/NOPB	Texas Instrument	1	1,31 €	1,59 €	1,59 €
Diodo LED	C503B-RCS-CW0Z0AA1	CREE	2	0,14 €	0,17 €	0,35 €
Resistencia 10K	10k 1/2w 5%	Generico	2	0,06 €	0,07 €	0,14 €
Resistencia 100K	100k 1/4w 5%	Generico	1	0,05 €	0,06 €	0,06 €
Resistencia 330	30 Ohm 1w 5%	Generico	2	0,11 €	0,13 €	0,26 €
Fotocélula Reflexiva	XUK9APANM12	Telemechanique	1	62,60 €	75,75 €	75,75 €
Alimentador Regulable	LM2596 HW-411	Generico	1	2,00 €	2,42 €	2,42 €
Relé	SRD-05VDC-SL-C	ELEC&Lifes	1	0,54 €	0,65 €	0,65 €
Transistor	PN2222ATA	ON SEMICONDUCTOR	1	0,13 €	0,16 €	0,16 €
Diodo rectificador	1N4004-E3/53	VISHAY	1	0,35 €	0,42 €	0,42 €
protoboard	PSG-BB-400	PRO-SIGNAL	1	2,22 €	2,69 €	2,69 €
batería 12V	AGM 12V-2.9AH MV1229	ENERGIvM	1	10,16 €	12,29 €	12,29 €
			TOTAL:	83,52 €	101,06 €	101,435

Tabla7.2: Presupuesto módulo Esclavo.

Lo cual deja un presupuesto total de coste de materiales de 227, 68 Euros.

MÓDULO	PRECIO SIN IVA	PRECIO CON IVA
Master	104,33 €	126,24 €
Esclavo	83,83 €	101,44 €
Total:	188,16 €	227,68 €

Tabla 7.3: Presupuesto total de los materiales del proyecto.

A este coste hay que añadirle el coste de mano de obra del diseño del prototipo, el cual asciende:

	Nº de horas	€/h	Total
Mano de obra	300	13	3900

Tabla 7.4: Presupuesto Mano de Obra.

La suma total de los materiales y la mano de obra asciende a 4.126,68€.

NOMBRE	PRECIO
Materiales	224,68 €
Mano de obra	3.900 €
Total	4.124,68 €

Tabla 7.5: Presupuesto Total del proyecto.

9. *Bibliografía.*

[1] *https://lastminuteengineers.com/nrf24l01-arduino-wireless-communication/*

[2] *https://howtomechatronics.com/tutorials/arduino/arduino-and-hc-05-bluetooth-module-tutorial/*

[4] *https://arduinobot.pbworks.com/f/Manual+Programacion+Arduino.pdf*

[5]*http://cienciesvirtuals.com/wp-content/uploads/2016/11/Arduino_Curso_Practico_de_Formacion.pdf*

[6] https://www.arduino.cc/en/Tutorial/HomePage

[7] https://forum.arduino.cc/

[8] https://blog.arduino.cc/

[9]https://files.pepperl-fuchs.com/webcat/navi/productInfo/doct/tdoct1797a_eng.pdf?v=23-MAR-20#:~:text=Dynamic%20mode%20can%20be%20used%20to%20detect%20moving%20objects.&text=The%20retro%2Dreflective%20sensor%20contains,swit%2D%20ching%20function%20is%20initiated.

[10] http://www.farnell.com/datasheets/1682238.pdf

[11]https://www.powerstream.com/battery-capacity-calculations.htm#:~:text=The%20key%20is%20to%20use,an%20inverter%20for%205%20hours.&text=Amp%2Dhours%20(at%2012%20volts,12%20%3D%20122.5%20amp%2Dhours.

Datasheet de los componentes utilizados.

[12] https://www.mouser.es/datasheet/2/357/OsiSense_XU_XUK9APANM12-1788293.pdf

[13] https://www.dena-de.com/uploads/shop/recursos/MV1229.pdf

[14]https://epow0.org/~amki/car_kit/Datasheet/ELEGOO%20UNO%20R3%20Board.pdf

[15]http://www.haoyuelectronics.com/Attachment/Mini-NRF24L01-SMD/nRF24L01P_Product_Specification_1_0.pdf

[16]https://components101.com/sites/default/files/component_datasheet/HC-05%20Datasheet.pdf

[17]https://www.ti.com/lit/ds/symlink/lm2596.pdf?ts=1599576159754&ref_url=https%253A%252F%252Fwww.google.com%252F

[18]https://www.ti.com/lit/ds/symlink/lm1117.pdf?ts=1599576196879&ref_url=https%253A%252F%252Fwww.google.com%252F

[19] http://www.datasheet.es/PDF/720556/SRD-05VDC-SL-C-pdf.html

[20] https://cdn.sparkfun.com/assets/learn_tutorials/4/2/3/1N4004.pdf

10. Anexo.

10.1. Plano y Fotografías 3D del Soporte para la sujeción del Equipo de Medida.

Para completar el Equipo se ha diseñado un soporte diseñado para posicionar en la pista de atletismo. Se ha utilizado una impresora 3D para su fabricación. En la figura 10.1 se detallan sus medidas.

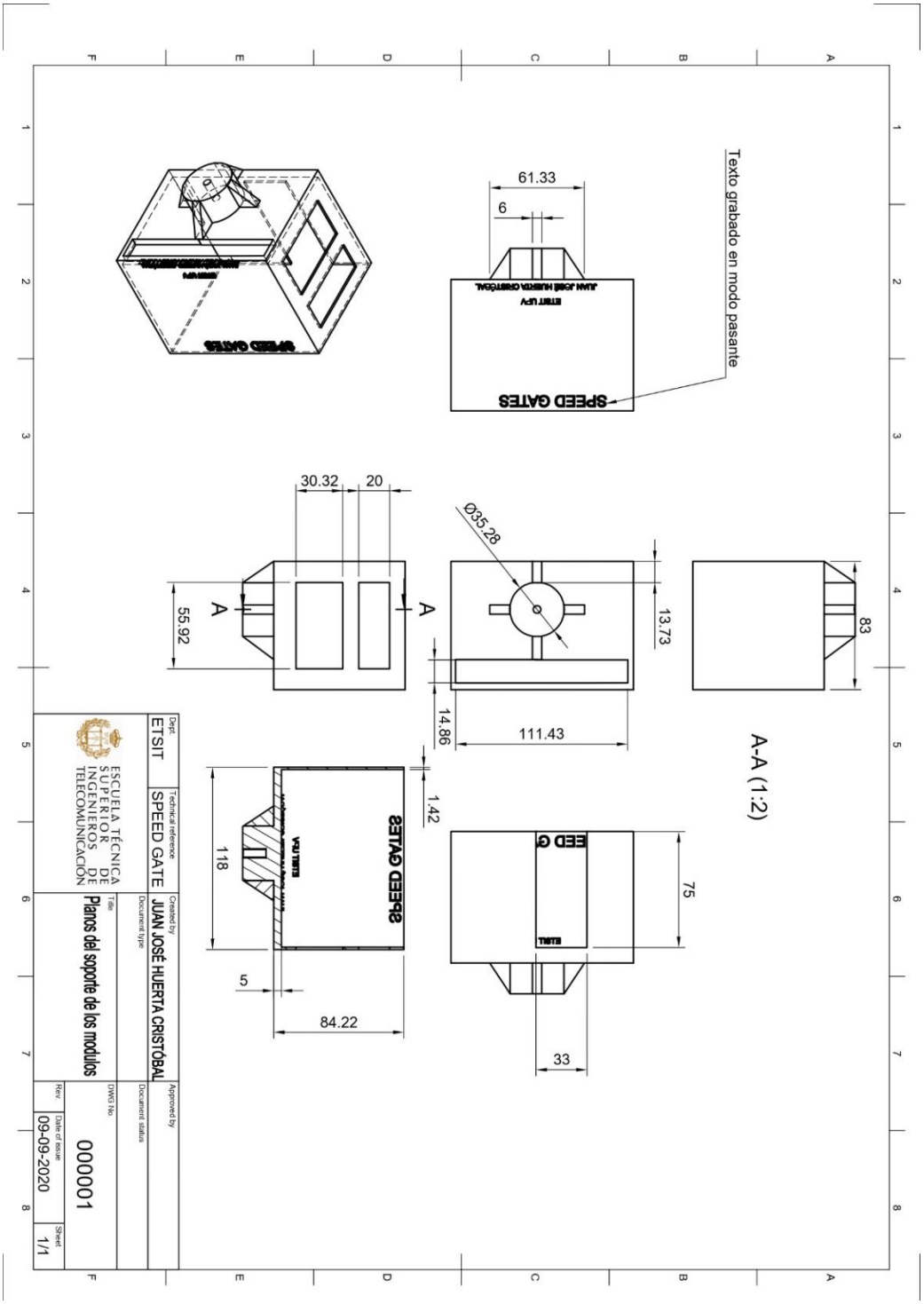

Figura 10.1: Plano de alzado, planta y vistas laterales del Soporte.

En las imágenes siguientes se muestran las fotos del Soporte obtenido mediante impresión 3D, el cual permite la posibilidad de anclar el Equipo de medida a un trípode fotográfico estándar

Figura 10.2: Vista frontal del Soporte para el módulo Master/Esclavo.

Figura 10.3: Vista Trasera del Soporte para el módulo Master/Esclavo.

Figura 10.4: Vista Inferior del Soporte para el módulo Master/Esclavo.

10.2. Fotografías de la PCB para el módulo Esclavo.

A continuación, se puede ver la placa PCB realizada en este proyecto para la inserción de los componentes necesarios para implementar el módulo del microcontrolador Arduino Esclavo.

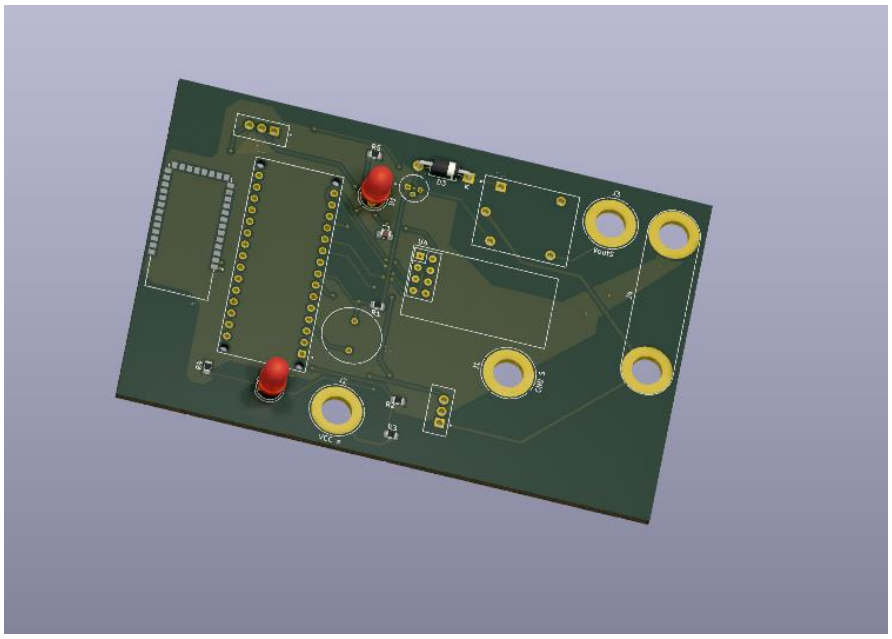

Figura 10.5: Cara superior de la PCB del módulo Esclavo

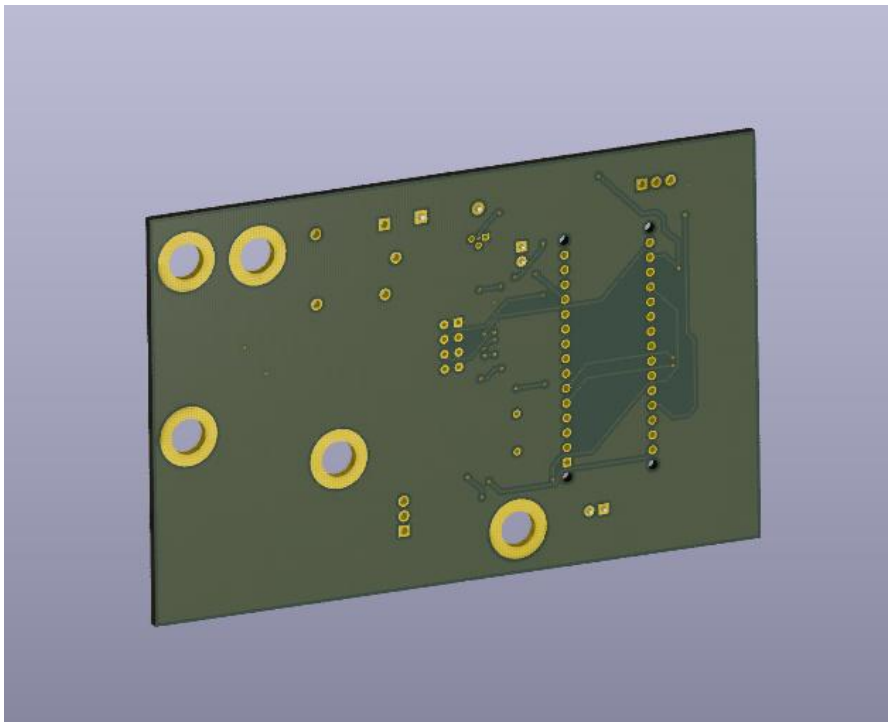

Figura 10.6: Cara inferior de la PCB del módulo Esclavo